IDIOT

IDIOT

TRABAJO FIN DE GRADO

CURSO 2024-2025

AUTORES:

PABLO GARCÍA FERNÁNDEZ (NOTA: 7.0)

ALEJANDRO LUQUE VILLEGAS (NOTA: 7.0)

TUTORES:

JUAN CARLOS FABERO JIMÉNEZ

JOSÉ LUIS VÁZQUEZ POLETTI

GRADO EN INGENIERÍA DE COMPUTADORES

FACULTAD DE INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

idIoT

idIoT

Trabajo de Fin de Grado en Ingeniería de Computadores

Autores:

Pablo García Fernández

Alejandro Luque Villegas

Tutores:

Juan Carlos Fabero Jiménez

José Luis Vázquez Poletti

Convocatoria: Septiembre 2025

Grado en Ingeniería de Computadores

Facultad de Informática

Universidad Complutense de Madrid

1 de Septiembre de 2025

Dedicatoria

A nuestras familias y amigos que nos han acompañado en esta etapa educativa. A Laura y a Cris por su apoyo incondicional.

AGRADECIMIENTOS

Este trabajo y estos estudios no se podrían haber realizado sin el calor de nuestras personas queridas. Por ello queremos agradecer a las distintas personas que han formado parte del proceso.

Queremos agradecer a nuestras familias, por estar ahí siempre durante todo este camino.

También agradecer a Javier y a Nacho, por todos los momentos compartidos durante estos años de carrera.

Queremos también acordarnos de nuestras parejas, Laura y Cris, por su apoyo incondicional y por animarnos a no rendirnos nunca.

Por último destacar a aquellos profesores que nos han motivado a aprender y desarrollarnos más en este mundo, en especial a Juan Carlos y José Luis, tutores de este trabajo. Sin sus apuntes, correcciones y apreciaciones, este trabajo no hubiera sido posible.

RESUMEN

Este Trabajo de Fin de Grado trata de señalar la falta de seguridad que existe en los distintos dispositivos IoT. Dichos dispositivos pueden ser comprometidos y afectar al entorno que las rodea, llevando consigo datos y accesos y pudiendo afectar al negocio en el que intervienen o a terceros. Los distintos estudios y noticias no dejan en buen lugar el tema de la seguridad, sobre todo teniendo en cuenta que en muchos casos las vulnerabilidades por las que se atacan los sistemas son ya conocidas con anterioridad, y basta con actualizar los componentes para evitar que esto pueda ocurrir.

Por ello, como parte del TFG, se ha realizado una herramienta que detecta las vulnerabilidades ya conocidas de los distintos componentes software que forman parte del sistema, generando un reporte del análisis de vulnerabilidades para ayudar al usuario o administrador del sistema a poder asegurar su sistema. Debido a la importancia que tiene Linux dentro de los sistemas operativos para dispositivos IoT, donde lideran este tipo de distribuciones, la herramienta está enfocada en este tipo de sistemas.

Se pretende, así, mejorar el entorno IoT con una ayuda clara para los distintos usuarios, gestores y fabricantes de estos dispositivos e invita a reflexionar sobre el crítico estado en el que se encuentra el mundo IoT en cuanto a seguridad se refiere.

Palabras clave: IoT, analizador, Python, Linux, vulnerabilidad

ABSTRACT

This end of degree thesis highlights the lack of security that exists in different IoT devices. Devices ranging from sensors to industrial machines can be compromised and affect the surrounding environment. These systems often contain crucial information, presenting a security risk to businesses and users. Studies and new articles display a concerning state of IoT security. This is exacerbated by the fact that many of these are previously known and can be solved many times with an update of the pertinent components.

Hence, the development of a tool that detects recorded vulnerabilities will be presented. Said tool also generates an analysis report to help the user, or system administrator, solve potential issues in the system. Our focus is a tool for Linux subsystems, as it is the leading operating system in IoT devices.

Ultimately, the goal is to improve IoT environments with a tool that helps users, administrators, and manufacturers of these devices while also providing a catalyst for further discussion on the state of security in these systems.

Keywords: IoT, analyser, Python, Linux, vulnerability

Índice de contenidos

ÍNDICE DE FIGURAS

Capítulo 1 - Introducción

Este capítulo explica la elección del tema del Trabajo de Fin de Grado, el objetivo que se persigue y la metodología de trabajo empleada.

1.1 Motivación

El internet de las cosas (IoT, por sus siglas en inglés, *Internet of Things*) es la red de dispositivos, llamados "cosas", que poseen software y componentes con el fin de comunicarse con otros dispositivos y sistemas a través de Internet[1]. Los dispositivos que se pueden considerar "cosas" van desde sensores, un frigorífico o una máquina de café, hasta grandes herramientas industriales como podría ser un brazo robótico, es decir, se encuentran en una gran variedad de ámbitos [1].

El término IoT fue acuñado por Kevin Ashton[2] en 1999. No obstante, antes ya había algún ejemplo de dispositivo IoT, como el de la máquina de Coca-Cola en la Carnegie Mellon University, que permitía conectarse a ella por medio de internet con el fin de averiguar si quedaban refrescos en la máquina, y evitar el desplazamiento para comprobarlo[3]. Actualmente, las mejoras tecnológicas de los sistemas inalámbricos, permite que el número de dispositivos conectados a internet haya aumentado, incluso existen neveras que hacen la lista de la compra [2].

El número de sectores que se benefician del IoT es enorme y muy variado: fábricas, sector público, atención sanitaria, transporte, etc. Esto demuestra la potencia de esta tecnología y el impacto que tiene en el mundo. Es reseñable hablar del IoT a nivel industrial, que permite mejoras en la seguridad y ciclo de fabricación. Al IoT Industrial se le conoce como IIoT. El IIoT son el conjunto de sensores, actuadores y dispositivos que trabajan y se comunican por medio de internet en el uso de

[1] https://www.oracle.com/es/internet-of-things/
[2] https://blog.avast.com/es/kevin-ashton-named-the-internet-of-things
[3] https://detri.epn.edu.ec/historia-iot/69

aplicaciones industriales, a diferencia del IoT que es para el uso de aplicaciones más generalistas[4]. Las aplicaciones que puede tener el IIoT van desde optimizaciones de rendimiento de la maquinaria hasta elementos para la disminución de accidentes industriales, ya sean laborales o generales.

Como vemos en la figura 1.1, para el 2019 ya había el mismo número de dispositivos IoT que de dispositivos no IoT conectados, y en 2024 había dos dispositivos IoT conectados por cada dispositivo no IoT [3].

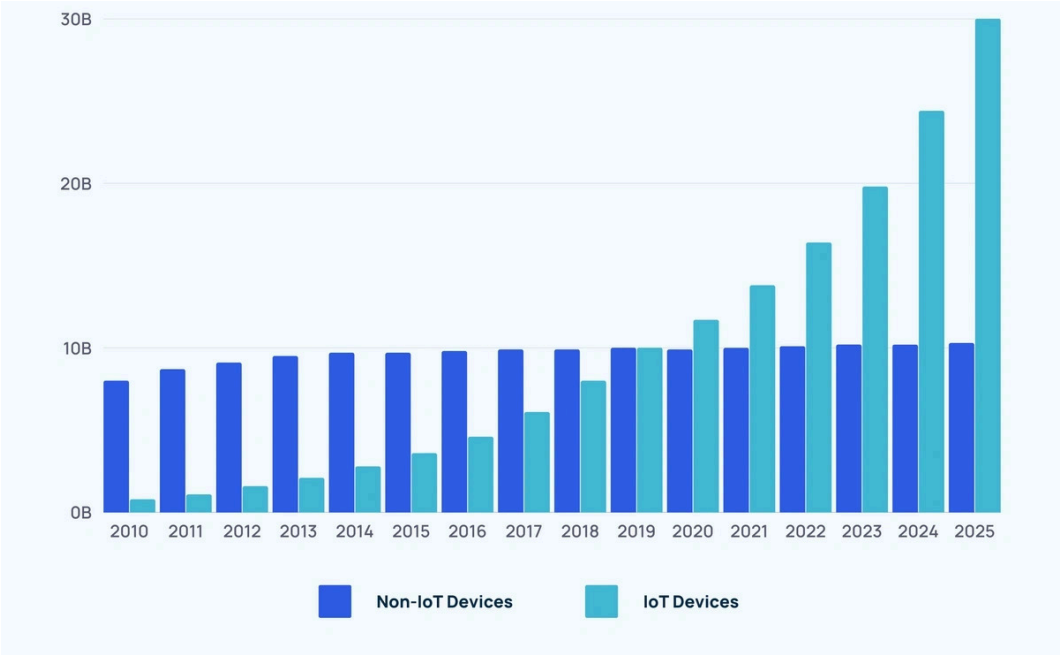

Figura 1.1. Número de conexiones globales activas. Fuente: Hybrid Efficient IDS Against Adversarial Attacks in IoT Networks

El nacimiento de esta tecnología ha proporcionado conectividad e integración a los distintos dispositivos que usamos en el día a día. Estos dispositivos son de bajo coste y de bajo consumo, por lo que tanto fabricantes como consumidores han apostado por ello. En la figura 1.2 se puede observar el crecimiento del mercado de dispositivos IoT y como se prevé la continuidad de este crecimiento durante los próximos años [4].

[4] https://www.iberdrola.com/innovacion/que-es-iiot

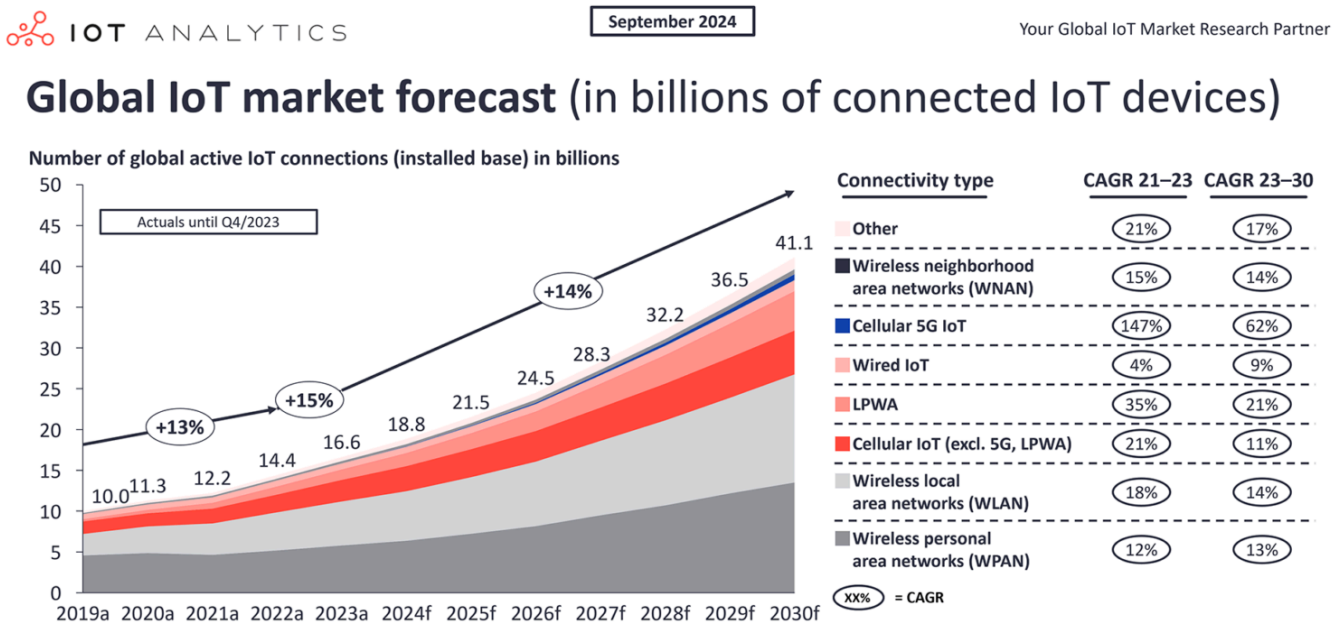

Figura 1.2. Predicción de mercado IoT global. Fuente: IoT Analytics

A pesar de los beneficios obtenidos; el rápido crecimiento del IoT conlleva riesgos de seguridad [5]. La priorización de un bajo consumo, el uso de hardware con menos potencia computacional, y la carrera hacia mercado a la que se ven sujetas las empresas y fabricantes provoca que estos dispositivos no destaquen, como decimos, por su seguridad. Por ello, muchos dispositivos IoT presentan vulnerabilidades en ámbitos como la autenticación, software desactualizado y puertos abiertos, entre otros [6]. Estas vulnerabilidades presentan una amenaza, pues pueden ser explotadas por los ciberdelincuentes con la finalidad de obtener datos, tomar el control de la máquina o incluso inhabilitarla. Esto puede suponer un problema para los consumidores que utilizan este tipo de dispositivos, agravándose en entornos críticos, como los industriales o empresariales, donde una vulnerabilidad puede tener consecuencias graves como la detención del proceso de negocio, filtración de información sensible o llegar a derivar en graves pérdidas económicas.

Contemplado el panorama que presenta el mundo IoT, es evidente la necesidad de mejora en las tareas de seguridad. Por ello pretendemos partir desde uno de los primeros componentes que forman parte del ciclo de desarrollo de los dispositivos IoT: el sistema operativo. En concreto, en este estudio se pretende evaluar los distintos componentes software de un sistema operativo IoT tipo Linux con un caso de uso y con la creación de una herramienta para este tipo de sistemas que, mediante el escaneo de los distintos componentes software detectados en el sistema, se señalen las vulnerabilidades ya conocidas para dichos componentes.

1.2 Objetivos

El objetivo principal de este trabajo es demostrar la vulnerabilidad de los dispositivos IoT a través de un análisis de seguridad por medio de un desarrollo de una herramienta realizada para este trabajo. El análisis se centrará mediante una demostración con un caso de uso, que consta de un sistema de tipo quiosco realizado sobre Ubuntu Core. Un sistema quiosco es un puesto en el que se pueden ver imágenes y video de manera que facilite algún servicio[5]. También pueden ser interactivos, como los sistemas de autoservicio comúnmente encontrados en establecimientos de comida rápida. El sistema quiosco que se quiere crear como banco de prueba es uno similar a las pantallas de noticias que se pueden encontrar en los buses, o las pantallas de información que se pueden encontrar en un centro comercial.

Con estos objetivos se pretende así realizar una reflexión de la falta de seguridad en los dispositivos IoT.

De esta manera se definen los siguientes objetivos específicos:

- Analizar el papel que tiene Linux en el mundo IoT.

[5]https://www.uptok.com/post/what-are-video-kiosks-and-how-are-they-transforming-retail-experiences

- Montaje y configuración de un sistema de tipo quiosco sobre Ubuntu Core.
- Desarrollar una herramienta que analice los componentes de un sistema Linux para poder detectar posibles vulnerabilidades que afecten a la misma.
- Generar análisis de seguridad de manera automática, haciendo reportes completos del sistema, donde hallar todas las vulnerabilidades que pueden afectar al sistema, destacando aquellas a las que el sistema es más vulnerable.
- Realizar un análisis demostrando la funcionalidad de la herramienta sobre el quiosco creado.

1.3 Plan de trabajo

Para la realización de este Trabajo de Fin de Grado se han seguido las siguientes fases:

- Fase 1: Investigación teórica y del estado actual del IoT.
- Fase 2: Investigación del impacto de Linux en el mundo IoT.
- Fase 3: Creación y configuración de un sistema tipo quiosco en Ubuntu Core, un Linux orientado a IoT.
- Fase 4: Desarrollo de una herramienta de detección de componentes software y vulnerabilidades asociadas a los mismos orientado a sistemas Linux IoT.
- Fase 5: Realización de pruebas de la herramienta sobre el sistema quiosco. Toma de datos y conclusiones.
- Fase 6: Desarrollo de la memoria.

Capítulo 2 - Estado de la cuestión

Este capítulo explica algunos conceptos necesarios para la comprensión de este Trabajo de Fin de Grado. También desarrolla la situación actual en la que se encuentra el tema tratado.

2.1 Formalización, demostración y explotación de vulnerabilidades

2.1.1 CVE

CVE, del inglés *Common Vulnerabilities and Exposures*, son las vulnerabilidades y exposiciones comunes. Estas son un conjunto de amenazas de seguridad detectadas que son públicas y pueden ser potenciales puntos de ataque para los dispositivos[6].

La lista de CVE incluye errores de código y errores de configuración que pueden permitir al atacante acceder o controlar el dispositivo, pudiendo realizar con el mismo cualquier tipo de actividad ilegal o no permitida.

Esta lista es muy beneficiosa tanto para desarrolladores como para clientes, ya que puede ayudar a prevenir o corregir fallos en el caso de los primeros, mientras que en el caso de los segundos ayuda a certificar la seguridad o falta de la misma para sus sistemas.

CVE incluye también para cada vulnerabilidad una puntuación que va del 0 al 10, siendo 10 la puntuación crítica[7]. La puntuación es un indicativo del riesgo que conlleva la existencia de dicha vulnerabilidad en un dispositivo. Por cada rango de puntuación reciben también un nombre que indica la gravedad[8]. Esos tramos son:

[6] https://www.fortinet.com/lat/resources/cyberglossary/cve
[7] https://www.bitsight.com/glossary/cve-rating
[8] https://docs.gitlab.com/17.5/user/application_security/vulnerabilities/severities/

- 0.0 – NONE, sin vulnerabilidad.
- 0.1 – 3.9 – LOW, riesgo bajo, fallos que pueden no ser directamente explotables. Ejemplo: Ausencia de configuración de seguridad de una *cookie.*
- 4.0 – 6.9 – MEDIUM, riesgo moderado, explotables con ciertas limitaciones, en conjunto pueden configurar un ataque mayor. Ejemplo: manejo incorrecto de sesiones HTTP.
- 7.0 – 8.9 – HIGH, riesgo alto, explotable e impacto significativo. Acceso parcial a los datos del sistema. Ejemplo: inyección XML.
- 9.0 – 10.0 – CRITICAL, riesgo máximo, deben ser investigadas en el mismo momento. Explotación fácil y de gran impacto que afectan directamente a todo el sistema y los datos en el mismo. Ejemplo: inyección SQL.

Esta puntuación se evalúa por medio de un sistema llamado CVSS, que actualmente se encuentra en su versión 4.0. Este sistema valora métricas que caracterizan a la vulnerabilidad, como por ejemplo el *Attack Vector* (AV), vector de ataque en inglés, o la *User Interaction* (UI), interacción del usuario. Estas están relacionadas con el impacto y la explotación de la vulnerabilidad. Por medio del cómputo de todas ellas se establece una puntuación[9].

Las métricas también sirven para crear el *string vector*. El *string vector* identifica, por medio de abreviaciones, cada una de estas métricas y el valor para la misma, separando métrica y valor por dos puntos. Para realizar la separación entre métrica y métrica se usa una barra inclinada. Además, al principio del vector se incluye un indicativo de la versión de CVSS que se está utilizando. Esto sirve para poder describir las características de la vulnerabilidad. Por ejemplo, la vulnerabilidad con *string vector* CVSS:4.0/AV:N/UI:A/… define que la vulnerabilidad tiene un *Attack Vector* con valor *Network*, que significa que el atacante la puede explotar remotamente, y una *User*

[9] https://www.first.org/cvss/v4-0/specification-document

Interaction con valor *Active*, lo que implica que para explotar la vulnerabilidad se necesita de la acción activa del usuario.

2.1.2 CWE

CWE, del inglés *Common Weakness Enumeration*, es una lista de debilidades comunes del software, fallos de diseño y desarrollo. De esta manera ayuda a los desarrolladores a identificar patrones que, de no ser detectados a tiempo, podrían generar vulnerabilidades en el sistema [7].

Es un elemento de seguridad que elimina fallos, de tal manera que aparte de mitigar potenciales vulnerabilidades, hace que la resolución del fallo sea mucho más barata y sencilla de realizar. Además, de esta manera la propagación del error es menor y la gravedad también disminuye. En la figura 2.1 vemos el impacto de los errores durante el ciclo de vida del producto software. Se observa que es menos costoso, en cuestión de esfuerzo y dinero, detectar el error previamente[10].

Figura 2.1. Impacto de vulnerabilidades y debilidades. Fuente: Mitre CWE

CWE tiene una fuerte relación con CVE, pese a sus distintos propósitos. Comprender ambas partes garantiza que se implementen medidas seguras para un funcionamiento seguro y correcto de los sistemas, tanto de manera proactiva, en la gestión de las debilidades, como de manera reactiva, en la gestión de las vulnerabilidades.

[10] https://cwe.mitre.org/about/index.html

Además, a las vulnerabilidades descubiertas se les suelen asociar las distintas debilidades que tengan relación con la misma, ayudando a la mitigación del problema por medio de la detención de causas raíz e implementando mejores medidas de protección. Por ejemplo, la vulnerabilidad CVE-2025-46577, tiene asociada el CWE-89, que es la debilidad relacionada con las inyecciones SQL, ya que la vulnerabilidad es una inyección SQL para Golden DB[11].

2.1.3 Shodan

Shodan[12] es un motor de búsqueda que indexa para sí los distintos dispositivos conectados a internet que tienen cualquier tipo de agujero de seguridad.

Por medio de esta herramienta se tiene acceso a diversos dispositivos, y es el mismo buscador el que te permite hacer un filtrado sobre el contenido, ciudad, etc. Todos los dispositivos que salen como parte de la respuesta son aquellos que no tienen políticas de seguridad aplicadas, como son los casos de las contraseñas por defecto.

Lo más atractivo de esta herramienta es que te permite acceder a una gran cantidad de información sobre un dispositivo sin hacer un previo estudio sobre qué vectores de ataque puede tener el mismo.

En la fecha 10 de febrero de 2025 se realizó una búsqueda en Shodan con la etiqueta IoT[13]. El resultado mostró 11.250 dispositivos encontrados en aquel momento con algún agujero en la seguridad. Esto plasma la falta de seguridad que hay en los dispositivos IoT ya que lo único que se ha necesitado ha sido una búsqueda de menos de un minuto.

Esta página podría incentivar al usuario a prestar más atención y proteger sus dispositivos. Esto es posible gracias a que la página incluye información como puertos

[11] https://www.cve.org/CVERecord?id=CVE-2025-46577
[12] https://www.shodan.io/
[13] https://www.shodan.io/search?query=IoT

abiertos, vulnerabilidades registradas en CVE, etc. Sin embargo, teniendo en cuenta la falta de empeño a la hora de proteger los dispositivos tanto por parte del fabricante como del usuario final, además del desconocimiento de estos últimos sobre la falta de seguridad en estos dispositivos, es muy probable que estos usuarios no investiguen si su dispositivo está registrado en Shodan.

2.2 Sistemas Operativos

Se llama sistema operativo al conjunto de herramientas y programas encargados de gestionar el hardware y software de un dispositivo. De esta manera tiene en cuenta la memoria, la CPU y los dispositivos de entrada y salida. Este elemento es necesario para aquellos dispositivos informáticos, sean del tipo que sean, para poder ejecutar las distintas tareas para las que esté enfocado. Permite la interacción con el usuario con el fin de poder realizar y personalizar las tareas que realiza, por medio de una interfaz, ya sea una línea de comandos o una interfaz gráfica [8].

Existen diversos sistemas operativos, donde de manera general los más conocidos al tener mayor cuota de mercado son Android, Windows, iOS, OS X y Linux[14]. Sin embargo, existen diversos sistemas operativos según el entorno y la necesidad del dispositivo y el uso que va a tener. De esta manera los tenemos distribuidos (con alta capacidad computacional), de PC (orientados al uso personal) móviles, multitarea y por lotes[15][9].

2.2.1 RTOS

Un sistema operativo en tiempo real, RTOS por sus siglas en inglés, es un sistema diseñado para ejecutar operaciones y procesar datos en un tiempo muy limitado. Teniendo esto en cuenta, se suele utilizar este tipo de sistemas en aquellos dispositivos

[14] https://gs.statcounter.com/os-market-share
[15] https://www.universitatcarlemany.com/actualidad/blog/tipos-de-sistemas-operativos/

donde la precisión es muy importante. Si el sistema se retrasa, puede quedar comprometido[16].

Estos tipos de sistemas tienen una programación determinista, es decir, están enfocados a las funcionalidades concretas de los dispositivos en los que corren. Es por ello que estos sistemas suelen usarse para sistemas embebidos y dispositivos IoT [10]. Todo esto es posible gracias a las características que rodean a los sistemas que usan RTOS, ya que cuentan con sistemas de archivos más básicos, bajo consumo de energía, memoria mínima y procesadores de bajo consumo. Por contraposición, la idea de realizar un sistema muy determinista, priorizando tareas y ejecutarlas en un espacio breve de tiempo, dificulta mucho el desarrollo en los sistemas RTOS, tardando más en realizarlos[17].

2.2.2 Linux y el mundo IoT

Linux es un kernel creado en 1991 por Linus Torvalds. Es de código abierto, es decir, el código fuente está disponible de forma gratuita. Su uso, junto con el proyecto GNU, formando GNU/Linux tiene diversos tipos de usos y fines, puesto que se usa en ordenadores personales, servidores web, supercomputadores y en dispositivos IoT entre otros, con usos principalmente en ámbitos como la programación o la ciberseguridad. Dentro de Linux existen las llamadas distribuciones, que son modificaciones al Linux maestro con el fin de crear un Linux nuevo, donde destacan Debian, Ubuntu o Arch Linux [18] [11][12].

Se trata de un sistema con muchas virtudes, entre las ya comentadas, como es el caso de que sea de código abierto, la personalización, y por ende flexibilidad, como

[16] https://www.universitatcarlemany.com/actualidad/blog/sistema-operativo-en-tiempo-real/
[17] https://www.seco.com/es/blog/detalles/rtos-vs-linux-embebido-una-guia-de-decision
[18] https://www.redhat.com/es/topics/linux/what-is-linux

otras como que presenta una fuerte seguridad, tiene buen rendimiento, es estable, tiene una comunidad muy activa, compatible con una gran gama de dispositivos hardware y con una gran conectividad[19] [13]. Estos últimos dos puntos le dan una gran ventaja para el mundo IoT. La conectividad es clave en sistemas compuestos por un gran número de dispositivos. Además, la compatibilidad permite que en las placas del mercado, y por ende en los dispositivos que se disponen en el mismo, se pueda contar con un sistema Linux para funcionar[20].

Según van creciendo los sistemas se les pretenden nuevos desarrollos. A la alarma del cuarto se la configura para algo más que suene y despierte. Se quiere también que se comunique con la cafetera para que empiece a poner a hacer el café. Y que le mande a una app las estadísticas de sueño. En este punto, distinguimos que los dispositivos IoT dejan de ser sistemas embebidos al uso, es decir, no funcionan en clave de una funcionalidad concreta. En este sentido se prefiere un sistema operativo a un RTOS, ya que aunque los RTOS cuentan con una gran precisión, su simplicidad y pocos recursos ante unos dispositivos en constante desarrollo, los dejan atrasados en la carrera de disponerse en dispositivos IoT al respecto de los sistemas operativos. Además, como ya se ha mencionado, el tiempo de desarrollo en el mundo RTOS es más largo, una desventaja en el mundo IoT, donde la evolución y desarrollo es constante[21].

Existen diversos sistemas operativos no RTOS que permiten hacer este tipo de integraciones. Y bien es cierto que existen otros productos como Windows 10 IoT, existen diversas distribuciones Linux que ratifican el poder que tiene este sistema para este mundo. Es también un tema de preferencia. En una encuesta realizada por Eclipse en el 2024 a los desarrolladores de sistemas IoT, Linux es el sistema operativo favorito

[19] https://tldp.org/HOWTO/Spanish-HOWTO-5.html
[20] https://github.com/linuxhw/HardwareSupport
[21] https://www.iotforall.com/linux-operating-system-iot-devices

para el desarrollo de dispositivos con capacidades limitadas en cuanto a memoria, procesamiento y potencia energética, con un 45%. También lidera en cuanto a los dispositivos de tipo *gateway*, que se encargan de comunicarse con otros dispositivos del internet de las cosas, gestionando y procesando datos de los mismos, y en los *edge nodes*, los dispositivos que actúan de nodos entre la red y los datos, donde la preferencia aumenta hasta el 67%. Esto demuestra el gran potencial que tiene Linux en este ámbito, donde los profesionales lo eligen en muchísimos proyectos, dado el potencial de sus virtudes ya mencionadas[22] [14][15][16][17].

Esa misma encuesta nos señala las distintas distribuciones de Linux que se usan para el mundo IoT. En este caso se encuentran distribuciones habituales, como Debian o Ubuntu, junto a otras más específicas para el uso IoT, como es el caso de Ubuntu Core y Raspbian.

2.3 Infraestructura y comunicación

2.3.1 Virtualización de sistemas

La virtualización es una tecnología que permite crear una representación del comportamiento de un servidor, un sistema de almacenamiento u otras máquinas. Se genera por medio de un software, el cual imita las funciones del hardware para varias máquinas. Este software puede gestionar varias máquinas virtuales a la vez[23].

El software encargado de abstraer y otorgar el hardware de la máquina local se le llama hipervisores [18]. Hay dos tipos de hipervisores:

- Hipervisor Tipo 1: son aquellos que se instalan dentro del hardware de la máquina, proporcionando un alto rendimiento al no necesitar de un sistema operativo que se encargue de gestionarlo, ya que el mismo hipervisor se encarga

[22] https://cribl.io/glossary/edge-node/
[23] https://aws.amazon.com/es/what-is/virtualization/

de otorgar y gestionar los recursos del hardware. Se utilizan en entornos donde la maximización de los recursos es esencial como los ámbitos de procesamiento de datos.

- Hipervisor Tipo 2: son aquellos softwares que se instalan sobre el sistema operativo de la máquina y por medio de la comunicación con el mismo obtiene los recursos que necesita. Estos hipervisores son los más conocidos de manera popular, ya que suelen ser los utilizados en ordenadores personales con softwares como VMWare o VirtualBox.

La virtualización nos da grandes ventajas, como el uso de distintos sistemas operativos de manera independiente entre ellos, la gestión de recursos, reducción de costes empresariales, escalabilidad y flexibilidad. Sin embargo, no es aconsejable el uso de las mismas si no se tienen conocimientos suficientes para poder configurarlas o si la máquina local se ve falta de los recursos necesarios para poder correr la virtualización [19].

2.3.2 SSH

SSH, Secure Shell[24] [25], es un protocolo criptográfico de red que proporciona un canal seguro sobre una red no segura. Por medio de un inicio de sesión se consigue que el usuario se autentique remotamente en un sistema, permitiendo el acceso a su contenido y a la ejecución de comandos. Este protocolo, con el 22 como puerto por defecto, cifra los mensajes entre cliente y servidor, protegiendo la comunicación de posibles interceptores y manipuladores.

De manera general, estas comunicaciones no se suelen autenticar por medio de un usuario y una contraseña, sino que suelen ir de la mano de lo que se llaman las claves SSH. Estas claves son generadas en parejas, una pública y otra privada, donde la

[24] https://www.vpnunlimited.com/es/help/cybersecurity/ssh
[25] https://www.cloudflare.com/es-es/learning/access-management/what-is-ssh/

privada queda en la máquina del cliente y la pública es enviada al servidor para poder cifrar la comunicación entre ambas máquinas.

2.4 Amenazas y ataques en sistemas

2.4.1 Ataque de denegación de servicios

Un ataque de denegación de servicios[26], DoS, por sus siglas en inglés Denial of Service, es un ataque por el cual el atacante tiene como objetivo interrumpir parcial o totalmente el correcto funcionamiento de un sistema. La técnica consiste en la sobrecarga de solicitudes desde el lado del atacante hasta provocar el fallo del sistema. Cabe destacar que este tipo de ataques se lanzan desde una única conexión, es decir, con un solo ordenador. En el caso de contar con múltiples conexiones se hablaría de un ataque de denegación de servicios distribuido (DDoS).

La manera en la que se deniega el servicio dependerá de la máquina atacada, sin embargo, suele producirse o por la incapacidad de atender la llegada masiva de paquetes enviados o por el gran consumo de recursos del sistema, ya sea CPU o memoria. Previo a llegar a este punto, el sistema suele comenzar a tener ciertos indicadores, como la pérdida de conectividad o un rendimiento bajo de la red, que van avisando de que se está sufriendo un ataque DoS.

2.4.2 Botnet

Una botnet[27] es un conjunto de ordenadores y dispositivos infectados por un malware de manera que pueden ser controlados remotamente por el ciberdelincuente. De esta manera, el atacante puede realizar actividades delictivas, por medio de parte o totalidad de la botnet, ya sean robos de credenciales, envío de spam, distribución de

[26] https://www.cloudflare.com/es-la/learning/ddos/glossary/denial-of-service/
[27] https://www.incibe.es/aprendeciberseguridad/botnet

malware o ataques DDoS. Este último tipo de actividad es por lo que se conoce de manera general a las botnets, ya que permite el ataque dirigido desde muchas máquinas al objetivo.

Debido al gran número de dispositivos IoT y la falta de seguridad de los mismos, este término está muy relacionado con el mundo IoT, porque los atacantes aprovechan la falta de seguridad y control de autenticación para hacerse con estos dispositivos para su propia botnet.

2.5 Programación y uso de APIs

2.5.1 Python

Python es un lenguaje de programación de alto nivel creado en 1991 por Guido van Rossum. Según el índice TIOBE, que lista los distintos lenguajes de programación según su popularidad, Python sería, con una diferencia de casi 17 puntos, el lenguaje de programación más popular, con un 26,14%, como se muestra en la figura 2.2. Esto tiene distintos motivos, como por ejemplo, que puede ejecutarse en distintos sistemas, que la sintaxis sea sencilla, similar al de la lengua inglesa, y que permite escribir programas en apenas unas pocas líneas de código. Además, cuenta con un gran número de librerías que permiten crear aplicaciones distintas según la necesidad. Dicho todo esto, no es de extrañar su popularidad y que se utilice en distintos ámbitos, como aplicaciones web, conectores de bases de datos y cálculos matemáticos complejos [28] [29] [20].

[28] https://www.w3schools.com/python/python_intro.asp
[29] https://www.tiobe.com/tiobe-index/

Aug 2025	Aug 2024	Change	Programming Language	Ratings	Change
1	1		Python	26.14%	+8.10%
2	2		C++	9.18%	-0.86%
3	3		C	9.03%	-0.15%
4	4		Java	8.59%	-0.58%
5	5		C#	5.52%	-0.87%

Figura 2.2. Top 5 de lenguajes de programación - August 2025. Fuente: TIOBE index

La simplicidad del desarrollo en este lenguaje y la compatibilidad con los distintos sistemas está favoreciendo el auge de este lenguaje en los sistemas IoT. En la encuesta de Eclipse hecha en 2024 a los desarrolladores del mundo IoT, Python era el tercer lenguaje favorito para los desarrolladores en cuanto a los dispositivos que cuentan con capacidades limitadas, con un 32%. También lideraba con un 39% en aquellos dispositivos que actúan de procesadores de datos, que se comunican con otros dispositivos IoT y gestionan red y datos haciendo de nodos entre ambas partes[30].

2.5.2 API

Una interfaz de programación de aplicaciones, API por sus siglas en inglés (*Application Programming Interface*), es un conjunto de reglas y protocolos que permiten a un par de aplicaciones comunicarse entre sí. Permite integrar fácilmente datos y funcionalidades de distintas fuentes, evitando así que el desarrollador las cree desde cero. De esta manera queda a disposición del desarrollador los datos que el desarrollador de la API crea necesario, ocultando aquellos detalles que sean sensibles o irrelevantes. Este tipo de mecanismos se ven en el día a día. Ya sea con la API de Google Maps, que ayuda a que las webs y aplicaciones de los negocios puedan indicar por

[30] https://outreach.eclipse.foundation/iot-embedded-developer-survey-2024

medio de la misma, la ubicación, o comparadores de viajes que comparan vía APIs distintos servicios para ofrecer el más barato para el usuario [21][22].

2.6 Actualidad

Bonaventura, Esposito y Bella (2023) [23] realizaron un estudio cuyo objetivo fue la investigación de las distintas vulnerabilidades que podrían ser detectadas en una bombilla IoT. Usaron lo que era la bombilla IoT más vendida en Amazon en la época que se estaba llevando a cabo la investigación, la Tapo L530E. La bombilla viene acompañada de una aplicación móvil en la que se controla las funciones que tiene, pudiendo hacerlo todo de manera remota. El estudio acaba encontrando cuatro vulnerabilidades que un atacante podría explotar si se encontraba cerca de la bombilla.

En primer lugar se encuentra una falta de autenticación entre la bombilla y la aplicación, lo que produce que el atacante pueda simular ser la bombilla durante la etapa de vinculación con la app. De esta manera, las credenciales de la aplicación y del Wi-Fi son obtenidas por el atacante. La segunda vulnerabilidad se encuentra en la fase de descubrimiento. Las credenciales en esta etapa eran cortas y expuestas y, por tanto, un atacante puede capturarlas y replicarlas, violando autenticación e integridad. La tercera tiene que ver con la encriptación. Aunque los mensajes iban cifrados, en una operación concreta se enviaba el mismo mensaje cifrado con la misma clave, permitiendo a un atacante utilizarlos. Esta vulnerabilidad aumenta su efecto mayor en conjunto con el cuarto problema: ni la bombilla ni la aplicación comprobaban el origen de los mensajes.

El resto del estudio se centra en cómo explotar estas vulnerabilidades y las soluciones que tenían para las mismas, reportando todo al fabricante.

El hecho de que este estudio se realizará con fines educativos, sin ánimo de violar ningún tipo de privacidad y en comunicación con el fabricante lo hace ver como una

mejora de seguridad ante un error que no se había tenido en cuenta. Sin embargo, esto no implica que siempre se actúe de esta manera.

Tras la lectura de noticias, se entiende que la seguridad de estos dispositivos no es tan robusta como debería.

Una de estas noticias trata el caso de una botnet que recibe el nombre de Mirai Botnet[31]. Esta botnet infecta dispositivos con un procesador ARC que ejecuta una versión de Linux reducida. Si se mantienen las contraseñas y usuarios por defecto, se ve sujeto a una posible infección, siendo así añadida a la botnet. La figura 2.3 ilustra un esquema donde se puede interpretar el funcionamiento de ataque de Mirai Botnet[32].

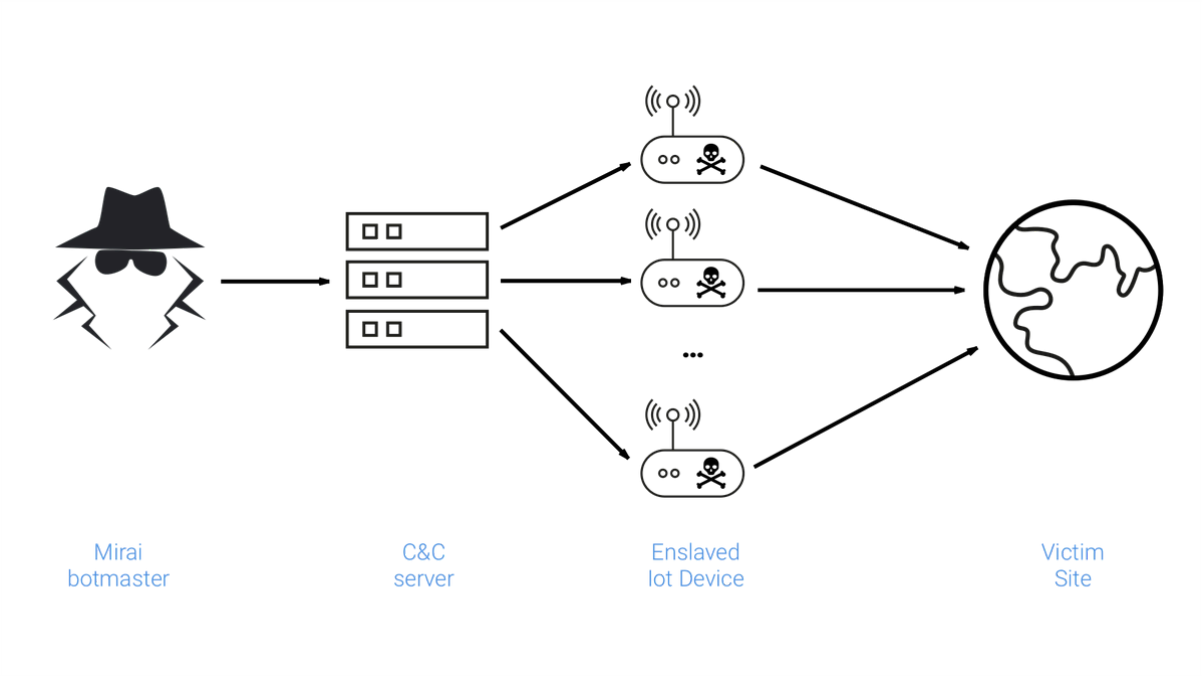

Figura 2.3. Módulo de ataque de la Mirai botnet. Fuente: Cloudflare

Este mismo 22 de enero de 2025, una botnet Mirai realizó un DDoS de 5.6 Tbps [24]. Este ataque usó 13.000 dispositivos IoT infectados. La noticia destaca que los dispositivos fueron infectados, posiblemente, mediante el uso de credenciales por

[31] https://www.cloudflare.com/es-es/learning/ddos/glossary/mirai-botnet/
[32] https://blog.cloudflare.com/inside-mirai-the-infamous-iot-botnet-a-retrospective-analysis/

defecto o de firmware sin parchear. Además, Cloudflare afirma que la fuerza de los últimos ataques hiper-volumétricos han hecho que suenen las alarmas en distintos sectores de la industria. Estos ataques son posibles gracias a miles de dispositivos IoT infectados.

En el estudio realizado por Bella y Bondi (2021) [25] se vuelve a mostrar la falta de seguridad que hay en los dispositivos IoT, en concreto, las impresoras. Los autores comentan que las impresoras, siendo un dispositivo muy común en las redes, tienen bajos niveles de seguridad. Este problema se ve acrecentado por el hecho de que muchos usuarios piensan que no merece la pena proteger estos dispositivos. A lo largo del artículo, se expone lo previamente dicho, evaluando el riesgo que pueden tener las impresoras, y experimentando con distintos ataques. Usando Shodan y el criterio de búsqueda puerto 9.100, usado para la impresión en red[33], encontraron miles de dispositivos potencialmente vulnerables.

Para estos posibles objetivos definieron una serie de posibles ataques. Estos eran:

- Registrar los dispositivos creando una botnet.
- Ataque DoS concreto para impresoras.
- Ataques a la privacidad por medio del acceso a los documentos que pasen por la impresora.

[33] https://www.cbtnuggets.com/common-ports/what-is-port-9100

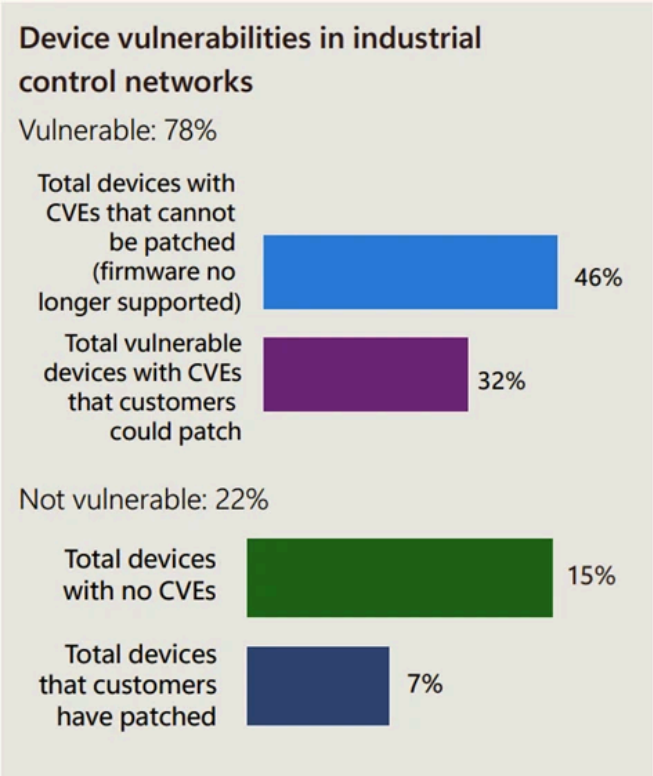

Figura 2.4. Estadísticas de dispositivos vulnerables en redes industriales de clientes. Fuente: Microsoft Digital Defense Report October 2023.

Los problemas de seguridad no afectan únicamente a dispositivos de particulares, también se ven afectados los dispositivos de empresas, fábricas y planta de agua. Una vez más, uno de los principales problemas son las "contraseñas débiles o software desactualizado con vulnerabilidades conocidas". La vulnerabilidad de estos dispositivos a ataques presenta una amenaza para la integridad y disponibilidad de infraestructuras críticas. La figura 2.4, obtenida del Microsoft Digital Defense Report (Microsoft, 2023) [26], se puede observar que un 78% de los componentes en redes de dispositivos de control son vulnerables.

En el informe también se revela que un alto porcentaje de las vulnerabilidades encontradas en dispositivos de redes de control ya son conocidas, convirtiéndose en un

foco de potenciales ataques dirigidos. Se destaca la presencia de vulnerabilidades en dispositivos de control industrial, trayendo consigo posibles consecuencias devastadoras para la operación de plantas y fábricas. En el sector de la salud, se han incrementado los ataques a dispositivos IoT, lo que pone en riesgo la seguridad de pacientes y la integridad de datos médicos [27].

En la manufactura de componentes electrónicos y automoción se ha incrementado el uso de robots, planteando un nuevo problema dado por la mayor interconexión que hay en las fábricas [28], ya que los robots presentan los mismos problemas de seguridad que los dispositivos IoT [29].

Cabe destacar que estos problemas en cuanto a seguridad no son solo técnicos, son también un problema de gestión o conciencia. Tanto usuarios como organizaciones no son conscientes de los riesgos que conlleva tener dispositivos de este tipo sin las medidas de seguridad necesarias. Por lo tanto, es de gran importancia que los usuarios estén concienciados sobre la relevancia de tener el software actualizado, las contraseñas por defecto cambiadas y utilizar proxies o VPN para conectar los dispositivos a la red principal.

En la actualidad, la seguridad de estos dispositivos sigue siendo un desafío. A pesar de que esta tecnología ha observado un avance, la presencia de las vulnerabilidades mostradas en los párrafos anteriores facilita la creación de botnets, ataques a gran escala y que se comprometan sistemas que manejan información sensible o confidencial.

Es fundamental destacar que la seguridad de los dispositivos IoT es una responsabilidad que recae en fabricantes y usuarios. Cada uno debe asumir su responsabilidad para la correcta implementación, implantación y mantenimiento de las redes de dispositivos IoT.

Lo que se demuestra con este apartado es la falta de seguridad que existe en el mundo IoT. El objetivo de este trabajo es complementar la información ya presentada,

creando una herramienta que permita analizar las vulnerabilidades de una máquina IoT que use Linux como sistema operativo. La herramienta creará un resumen de las vulnerabilidades encontradas y de qué gestor de paquetes o ruta de ejecutables proviene. Se ha decidido esta ruta, ya que los sistemas operativos son una parte fundamental de los dispositivos IoT, puesto que deben ser eficientes en rendimiento y recursos.

Capítulo 3 - La herramienta

En este capítulo se comentará el desarrollo de la herramienta desarrollada para detectar vulnerabilidades y el montaje del entorno para realizar pruebas de la misma.

3.1 Introducción y objetivo

Con el fin de poder demostrar el problema de seguridad que presentan los dispositivos IoT, se desarrolló una herramienta encargada de detectar los distintos componentes software que forman parte del mismo y consultar por las vulnerabilidades conocidas presentes en él. Esta consulta se realiza mediante la API de la *National Vulnerability Database* (NVD), repositorio del gobierno de los Estados Unidos encargado de recopilar vulnerabilidades, estándares de seguridad y el cumplimiento de los mismos[34] [35].

El enfoque de esta herramienta está orientado hacia sistemas operativos de tipo Linux, dada la importancia que tienen para el mundo IoT, como se comentaba en el capítulo anterior.

De esta manera, el usuario de este analizador puede conocer las distintas vulnerabilidades de los componentes del sistema y poder remediarlas.

3.2 Detección del software

Para la creación de esta herramienta ha sido importante conseguir los distintos componentes del sistema, la detección del software. Dentro de este ámbito se distinguen dos líneas distintas de detección: detección de software proveniente de distintos gestores de paquetes y detección de software extraído de ejecutables del sistema operativo o paquetes de *snap*.

[34] https://nvd.nist.gov/developers/vulnerabilities
[35] https://www.nist.gov/programs-projects/national-vulnerability-database-nvd

3.2.1 Software proveniente desde distintos gestores de paquetes

En los sistemas operativos de tipo Linux existe un término que hace referencia a la manera de distribuir los programas. Este término es *paquetes*, que conjuntamente a sus gestores, encargados de instalarlos, actualizarlos y borrarlos, permiten una mejor distribución y mantenimiento del software. Todo esto se consigue mediante repositorios de software que almacenan estos paquetes de manera que puedan ser accesibles desde el sistema operativo [30].

Existen diversos gestores de paquetes, ya que cada distribución o familia de ellas dispone del propio. Esto se debe a requisitos y formas de trabajo distintas de las distintas distribuciones. De esta manera, un mismo sistema Linux puede tener varios gestores de paquetes, sin embargo, esto no garantiza su funcionamiento, puesto que cada sistema tiene el suyo concreto que funciona según se le requiere[36].

Por ende, con el fin de poder detectar software desde gestores, se han dispuesto en la herramienta consultas a estos, con el fin de poder detectar el sistema y su gestor e incluso algún gestor más si estuviera instalado en también en el sistema. De cada uno de estos gestores se obtiene la información necesaria para consultar a la API y poder identificar aquellas vulnerabilidades en el sistema. Los gestores de paquetes analizados son: *snap*, *dpkg*, *apt* y *pacman*.

3.2.2 Software extraído de ejecutables del sistema operativo

También se extraen los ejecutables que hay en el sistema. Durante el desarrollo de la herramienta fue necesario incorporar esos ejecutables, ya que no son dependientes de ningún gestor de paquetes y también son relevantes. De hecho, en estudios previos

[36]https://umh2820.umh.es/unidades-didacticas/gnu-linux/instalacion-de-diferentes-distribuciones-de-linux-y-gestion-de-sus-paquetes-de-software/

identificamos vulnerabilidades en OpenSSH, ejecutable que no va a través de ningún gestor en el sistema estudiado, por lo que consideramos relevante incluirlo.

Se amplió esta funcionalidad a las dependencias de cada paquete tipo *snap*, ya que estos al ser autocontenidos para que funcionaran de manera independiente, contaban en el directorio donde habían sido instalados con aquellas dependencias necesarias para que funcionaran[37].

Gracias al uso de los comandos find y ls, dentro de estos directorios, encontramos las carpetas /usr/bin/ con los ejecutables de cada paquete.

Para obtener la información de cada ejecutable, fuera del sistema o dependencia de un paquete snap, se realizan pruebas con los distintos flags que sacan la información de la versión, y en el caso correcto, se usan expresiones regulares para extraer esa información que se muestra. Con todo esto, se obtiene lo necesario para poder construir el CPE.

3.3 Estructura del código del analizador

A continuación se comenta el código del analizador, escrito en Python. Puede verse al completo en el siguiente repositorio, en la carpeta *scripts*:

https://github.com/pagarc24/tfg_idiot

Primeramente, el código se encargará de recoger todos los componentes del sistema. Para ello se cuenta con una función llamada *collector* que irá recogiendo los componentes de cada uno de los gestores de paquetes ayudándose de las funciones auxiliares. También hará lo mismo con los ejecutables ayudándose de una función auxiliar. En cada función recoge los datos de nombre, de versión, y en algunos casos, del creador. Además, con esos datos se crea el CPE, por medio de *cpe_constructor*, una

[37] https://snapcraft.io/about

manera de nombrar e identificar el componente, esencial para realizar la consulta sobre la API de NVD[38]. Este está compuesto principalmente por el nombre del producto, el fabricante y su versión con el fin de actuar como identificador único del producto. De esta manera resulta más sencillo hacer las consultas de cada componente.

Una vez recogidos los datos se mandaban uno a uno los componentes a consulta con la API de NVD. En ella se consultaba por las vulnerabilidades de cada componente y se creaba un reporte. Cada reporte creado para cada componente, mediante la función *component_analysis*, se iba incluyendo en el fichero que actúa de reporte final con todos los componentes analizados. Para un mejor rendimiento se incluyó una clave facilitada por NVD, de manera que las consultas sobre NVD eran más rápidas. Por cada vulnerabilidad, además, se evaluaba si tenía una serie de condiciones que la hacen significante, por medio de la función *must_be_highlighted*, y de ser así se incluía en el reporte final destacándola, a la par que se almacenaba para ser mostrada una vez terminase la ejecución.

Finalmente, se indicaba al usuario las vulnerabilidades destacadas y el criterio por el cual se han destacado, en qué fichero puede ver el reporte completo y un pequeño resumen de lo encontrado.

Para poder usar el analizador, el sistema debe contar con Python 3 instalado. También debe hacer uso de unas cuantas librerías, algunas ya incluidas según se instala Python. Las otras necesarias, *distro* y *requests*, se instalan mediante un script llamado setup.sh. Este script también está en el repositorio, en la misma carpeta que el analizador. El script se encarga de preparar el entorno e instalar las librerías necesarias.

Se eligió Python para el desarrollo de este analizador por tres razones principales:

[38] https://nvd.nist.gov/products/cpe

1. La librería *requests* permite realizar consultas a la API de manera sencilla y pudiendo evaluar las respuestas con facilidad.
2. Python cuenta con gran popularidad dentro del mundo IoT, como se vio anteriormente en los conceptos.
3. Python es un lenguaje de programación multiplataforma, donde en los entornos Linux llega a estar presente en muchas distribuciones, y contando con documentación para instalarlos en aquellas que no cuentan con el mismo[39].

3.4 Demostración

3.4.1 ¿Qué es Ubuntu Core?

Ubuntu Core es una versión del sistema operativo Ubuntu que está diseñado específicamente para sistemas empotrados y dispositivos IoT. Por medio de él se pueden desarrollar sistemas concretos para distintos componentes hardware. Este sistema está presentado para ser usado en entornos en los cuales la seguridad y la fiabilidad son fundamentales[40].

Su principal característica es que cada elemento del sistema corre en un espacio confinado por medio de un entorno aislado, técnica conocida comúnmente con *Sandboxing*, asegurando el correcto funcionamiento del sistema ante posibles fallos, ya que estos solo afectarían al elemento en el que se ha generado. Esto se consigue por medio de paquetes tipo *snap*[41].

Los paquetes tipo *snap*, formato desarrollado por Canonical, la empresa detrás de Ubuntu, empaquetan las aplicaciones junto a sus dependencias. De esta manera, por

[39] https://docs.python.org/es/3.8/using/unix.html
[40] https://ubuntu.com/core
[41] https://documentation.ubuntu.com/core/explanation/core-elements/snaps-in-ubuntu-core/

medio de esta encapsulación permite una instalación y ejecución en nuestro sistema sin tener conflictos. Estos paquetes brindan además compatibilidad entre sistemas, aislamiento, seguridad, actualizaciones automáticas y el uso de múltiples versiones de manera simultánea. Además, su uso es sencillo, porque por medio de simples comandos se puede realizar una fácil y buena gestión del software instalado.

Ubuntu Core establece un inicio de sesión por medio de certificados SSH, los cuales son muy seguros. De esta manera, se establece en la configuración del sistema el usuario que va a utilizar dicho sistema por medio del correo electrónico asociado a su cuenta de Ubuntu One. Además, para certificar que es el mismo deberá conectarse por medio de su clave privada, habiendo previamente añadido su clave pública a su cuenta de Ubuntu One. Así se asegura que el sistema solo está permitiendo el acceso a un posible usuario, aumentando la seguridad debido a eso mismo.

Canonical destaca cuatro principales características de su sistema operativo orientado a IoT que le hacen tener el conjunto completo de herramientas para poner a funcionar el sistema del dispositivo. Estas son:

1. Seguridad continuada, "creando el sistema empotrado Linux más seguro hasta la fecha", con un compromiso de soporte y distintas actualizaciones de 12 años.
2. Gestión del dispositivo, dispositivos accesibles y confiables siempre actualizados por medio de actualizaciones OTA. Las siglas OTA vienen de *Over-The-Air*, que significa a través del aire, y es una forma de actualizar el software o el firmware de los sistemas inalámbricamente[42].
3. Se organiza fácilmente por contenedores, de manera rápida y segura, permitiendo a los componentes, kernel, sistema operativo y aplicaciones estar separados y confinados[43]. Esto se consigue por medio de los anteriormente

[42] https://consumer.huawei.com/es/support/faq/que-es-una-actualizacion-ota/
[43] https://snapcraft.io/iot

mencionados paquetes tipo *snap*, los cuales brindan al sistema protección para que nada se corrompa por medio de las actualizaciones independientes.

4. Tienen un fuerte ecosistema hardware. Esto lo consiguen gracias a la comunicación con los distintos fabricantes de los dispositivos y proveedores de silicio, construyendo conjuntamente el sistema y probándolo constantemente.

Dadas estas características, Canonical identifica cinco casos de uso donde recomiendan enormemente sus sistemas:

1. Publicidad y venta digital, por medio de pantallas con anuncios o pantallas táctiles que permiten al usuario elegir el producto deseado. El hecho de que Ubuntu Core sea compatible con las principales placas del mercado[44] (como Raspberry Pi, Intel NUC, Dragonboard) reduce el *time-to-market*, ya que se ahorra tiempo de desarrollo y se evitan problemas de integración. Posteriormente, se desarrollará un poco más este punto explicando lo que es un sistema de tipo quiosco.
2. Ciudades inteligentes, brindando automatización e innovación a áreas urbanas como coches autónomos o la ya mencionada publicidad digital. Son servicios que gracias a Ubuntu Core les permite ser distribuidos, evitando errores en cadena. Además, cuentan con una fuerte seguridad. Este uso está avalado por empresas como Telefónica, Bosch y AT-T entre otras[45].
3. Robótica, donde Ubuntu Core, por medio de snap, permite integrar el desarrollo del código y optimizar su uso en este tipo de sistemas con todas las ventajas de seguridad, compatibilidad y de gestión anteriormente mencionadas. Se puede observar en la figura 3.1 cómo sería el proceso de desarrollo de elementos de

[44] https://documentation.ubuntu.com/core/reference/system-requirements/
[45] https://ubuntu.com/core/stories

robótica con las herramientas de Canonical, donde Ubuntu Core terminaría de ensamblar el sistema diseñado para ser cargado en el hardware concreto[46].

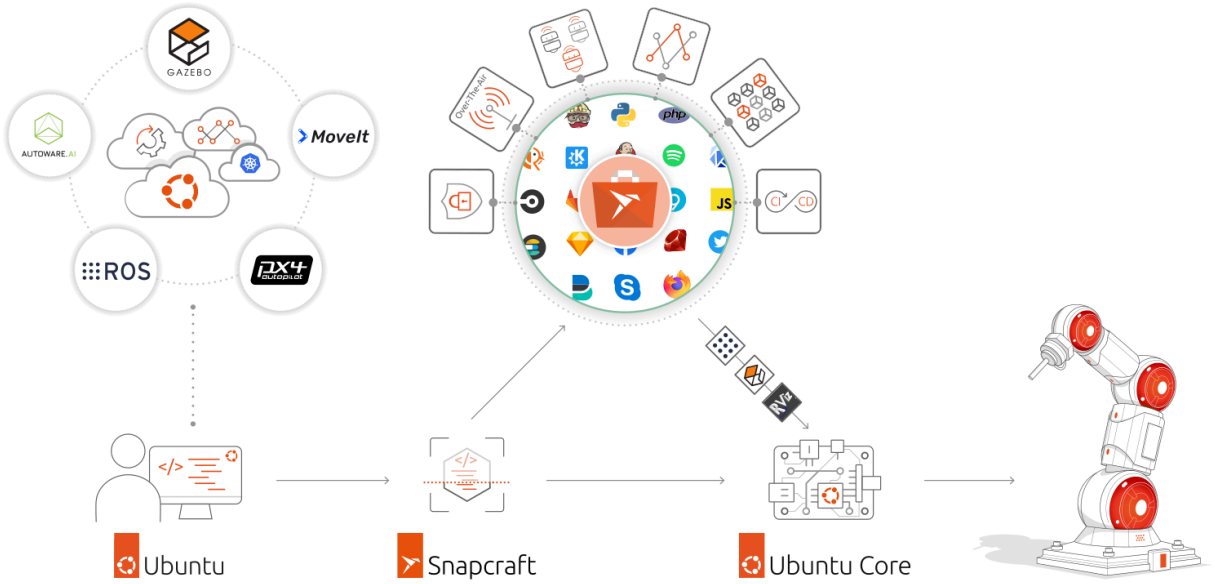

Figura 3.1. Diagrama de flujo del desarrollo de robots con herramientas de Canonical. Fuente: Ubuntu Robotics.

4. Industria, donde el uso de Ubuntu Core en las placas de los robots ha permitido que el sistema forme parte de las soluciones de las empresas industriales. Así es como Canonical, la ya mencionada empresa encargada de Ubuntu Core, forma parte de ctrlX Automation, una plataforma industrial abierta de Bosch para la automatización de los procesos de desarrollo. La plataforma integra hardware y software en sí misma. El uso de Ubuntu Core en esta plataforma trae consigo sus principales ventajas, trayendo un sistema seguro, compatible y distribuido, permitiendo el desarrollo de PLCs con la cultura de *snaps*. Esta incorporación le

[46] https://ubuntu.com/robotics

ha permitido a Bosch bajar un 20% los costes de ingeniería de la plataforma[47] [31].

5. Industria de la automoción, donde el sistema interviene en el desarrollo de coches autónomos y en los procesos de fabricación, mejorando el producto y el modelo de negocio. Son muchas las empresas de este sector que cuentan con Canonical para el desarrollo de sus productos, ya sean Toyota, Volkswagen, Mercedes, Ford, Skoda, …

Ubuntu Core forma parte de los productos de la familia Ubuntu, que a su vez vienen de la distribución Debian [32]. Según la ya mencionada encuesta hecha por Eclipse a los desarrolladores IoT, los sistemas nacidos de Debian son líderes en uso. Ubuntu Core registró un 10% de preferencia, sin embargo, se eligió por dos motivos principales:

1. Este sistema tiene un amplio esparcimiento en el mundo IoT. Además, como se ha comentado anteriormente, muchas empresas confían en él. Es por ello que usar este sistema como prueba de lo vulnerables que son los sistemas IoT por medio de la herramienta señalan el daño potencial que haría a las distintas industrias la falta de seguridad en los sistemas IoT.
2. Sistema de paquetes tipo *snap*. Como ya se ha comentado, Ubuntu Core utiliza este gestor de paquetes. Además, es un gestor que se acopla bastante bien a los distintos sistemas y que casa con la filosofía de uso de los dispositivos IoT, ya que permite aislar los paquetes entre sí, pudiendo gestionarlos de manera autónoma e irlos quitando o poniendo según la necesidad del sistema, sin que esto interfiera con los demás paquetes. *Snap* no es el único gestor de paquetes que se analiza con esta herramienta, sin embargo, sirve como un buen punto de

[47] https://apps.boschrexroth.com/microsites/ctrlx-automation/en/portfolio/

partida para detectar y analizar software debido a la popularidad en el mundo IoT [33].

3.4.2 Montando un sencillo sistema de tipo quiosco

¿Qué es un sistema de tipo quiosco?

Se refiere a *quiosco* a aquellos paneles interactivos que mejoran la experiencia de usuario. Sus usos pueden ser variados, desde máquinas de autoservicio hasta mapas interactivos del recinto que ocupan[48].

Debido a que requieren una constante conectividad, ya sea para mostrar nueva información del lugar o detalles de pago, es natural ver a los sistemas de tipo quiosco como dispositivos IoT, donde la potencia de este mundo y su constante evolución han mejorado la eficiencia y la experiencia de uso[49].

Paso a paso: montando un quiosco en Ubuntu Core

A continuación se dan los pasos que se siguieron para la construcción del quiosco. Se pretendía realizar un sistema simple, ya que el objetivo de este trabajo no era la realización del mismo. Es por ello que se eligió simular un carrusel de imágenes, actuando como una pantalla de noticias similares a las que se ven en el transporte público [34].

El montaje se realizó sobre el software de virtualización QEMU, en una máquina Debian 12. QEMU[50] es un software con licencia *open source* dedicado a la emulación y virtualización de máquinas y sistemas. Además, es un hipervisor de tipo 2, sin embargo, se puede identificar que en esta virtualización llega a comportarse como si fuera de tipo 1. Esto es bastante conveniente, ya que el sistema que se cree a partir de

[48] https://www.getguru.com/es/reference/kiosk-software
[49] https://www.iotforall.com/empowering-automated-retail-with-advanced-iot-connectivity
[50] https://wiki.qemu.org/Main_Page

Ubuntu Core actuará directamente sobre la máquina, por lo que nos interesa que tenga un rendimiento similar al de trabajar sobre el hardware como si se tratara directamente de un dispositivo IoT.

1. Conseguir la imagen de Ubuntu Core. En este caso, se eligió la última versión disponible, la 24, con arquitectura amd64. Las imágenes quedan disponibles en la página oficial en el apartado *Testing Platforms*[51].
2. Instalar QEMU. En el caso de la máquina Debian se realizó por medio de snap de la siguiente manera:

   sudo snap install qemu-virgil

   sudo snap connect qemu-virgil:kvm

3. Registrarse en Ubuntu One. De esta manera se podrá autenticar el usuario en Ubuntu Core. Una vez registrado deberá almacenar en la cuenta la clave pública SSH con la que se pretende iniciar sesión. En la figura 3.2 se muestra cómo se genera el par de claves SSH y en la figura 3.3 cómo se pueden adjuntar a la cuenta de Ubuntu One.

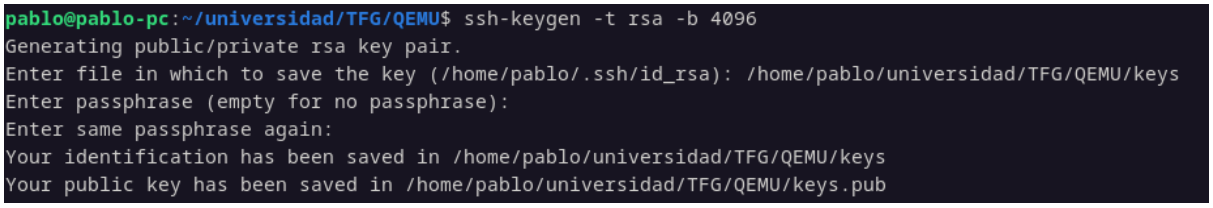

Figura 3.2. Generación de claves SSH.

[51] https://documentation.ubuntu.com/core/reference/testing-platforms/

Figura 3.3. Importando llaves a Ubuntu One. Fuente: Ubuntu One.

4. Una vez puesta la clave, se puede lanzar la virtualización, como se muestra en la siguiente imagen. Entre otras configuraciones, cabe destacar que se redirige el puerto de comunicación SSH, clave para el inicio de sesión y comunicación en este sistema. Es por ello que se enlaza el puerto 22 de la virtualización con el 10022 de la máquina desde la que se lanza QEMU. En la figura 3.4 se muestra el comando para lanzar dicha virtualización.

```
pablo@pablo-pc:~/universidad/TFG/QEMU-VIRGIL$ qemu-virgil -enable-kvm -m 16384 -device virtio-vga,virgl=on\
 -display sdl,gl=on -netdev user,id=ethernet.0,hostfwd=tcp::10022-:22\
 -device rtl8139,netdev=ethernet.0\
 -drive file=/snap/qemu-virgil/current/usr/share/qemu/edk2-x86_64-code.fd,if=pflash,format=raw,unit=0,readonly=on\
 ubuntu-core-24-amd64.img
```

Figura 3.4. Iniciando Ubuntu Core con QEMU.

5. La primera vez que se virtualiza, Ubuntu Core muestra un menú para configurarlo, como el que se muestra en la figura 3.5. Allí se indica el correo asociado a la cuenta de Ubuntu One, de manera que a la hora de acceder se puedan usar las claves SSH, ya asignadas a esa cuenta.

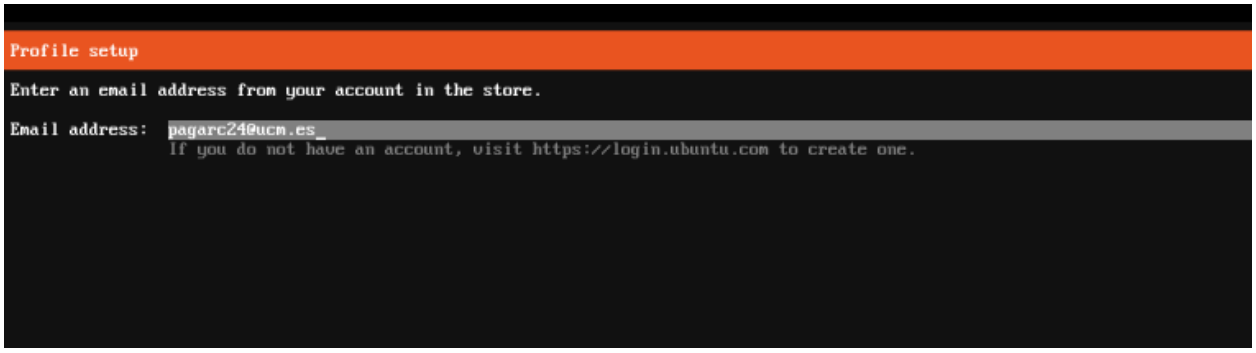

Figura 3.5. Inicio de perfil en Ubuntu Core.

6. Llegados a este punto se muestra un mensaje indicando que se puede conectar vía SSH al sistema. Pese a las indicaciones dadas por el mensaje mostrado, se debe usar el siguiente comando (donde *pagarc24* es la cuenta usada en Ubuntu One, y keys es la clave privada), ya que se trata de una virtualización:

   ssh -v -i keys pagarc24@localhost -p 10022

7. Ahora ya dentro del sistema se configura el quiosco. Primeramente, debemos instalar vía snap los paquetes *mir-kiosk* y *wpe-webkit-mir-kiosk.*
8. Se crea una carpeta donde poner todo el contenido del carrusel. En ella se crea un fichero index.html que será el que muestre el quiosco. Este fichero incluye un script en Javascript que llama a una función que se encarga de cambiar la imagen del carrusel cada 15 segundos. El código de index.html es el siguiente (se presupone que ya se ha conseguido introducir en la carpeta los ficheros de imágen que se mencionan):

```
<!DOCTYPE html>
<html>
<body style="margin:0;overflow:hidden;background:black;">
<img id="animation" src="imagen1.png" alt="Noticias del día">
<script> var images = ["imagen1.png", "imagen2.png", "imagen3.png", "imagen4.png",
"imagen5.png"];
var index = 0;
function playAnimation() {
                        var img = document.getElementById("animation");
                        img.src = images[index];
                        index = (index + 1) % images.length;
                        setTimeout(playAnimation, 15000);
}
                playAnimation();
</script>
</body>
</html>
```

9. Por medio del módulo http.server de Python se lanza un servidor en local en el puerto 8080: python3 -m http.server 8080
10. Se indica al paquete wpe-webkit-mir-kiosk la dirección que debe mostrar, que debe ser la creada en el paso anterior. Una vez realizado se resetea el paquete para cerciorarse de que todo funcione correctamente. Los comandos son los siguientes:

    sudo snap set wpe-webkit-mir-kiosk url=http://localhost:8080

    sudo snap restart wpe-webkit-mir-kiosk

Realizados todos los pasos, el quiosco queda creado y se muestran las noticias en la pantalla.

3.4.3 Ejecución: resultados y conclusiones

La herramienta analizó el sistema de Ubuntu Core con los resultados que aparecen en la figura 3.6, destacando aquellas vulnerabilidades que se califican como importantes y haciendo un reporte completo en un fichero.

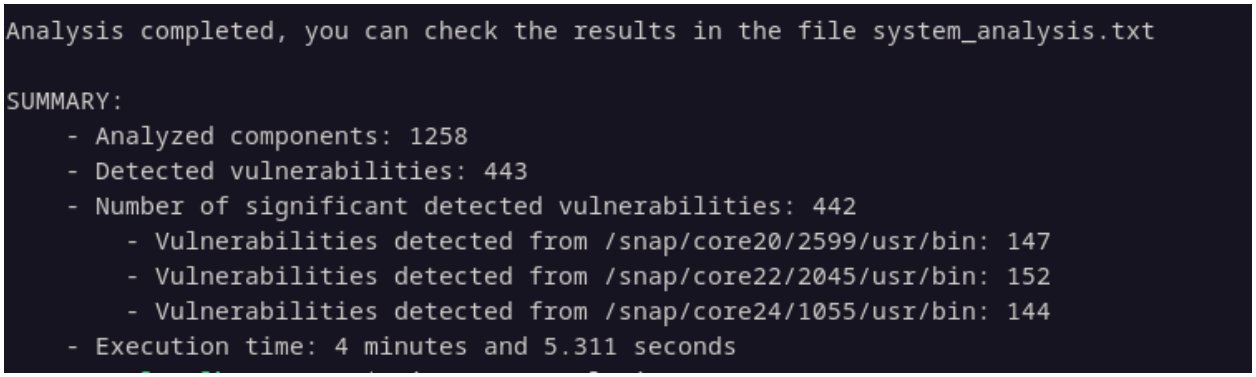

Figura 3.6. Resumen del analizador.

Dentro del reporte completo aparecen también las fuentes de donde se han sacado los componentes, como se muestra en la figura 3.7, buscando en los gestores de paquetes y en todos aquellas rutas de ejecutables que se hayan encontrado. Adicionalmente, se encuentra el desglose de cada componente y las vulnerabilidades, de tenerlas, que se encuentren del mismo. Esto se puede ver en la figura 3.8 , donde se muestra un extracto de un componente con vulnerabilidades conocidas y la información de las mismas, y en la figura 3.9 un componente sin vulnerabilidades conocidas.

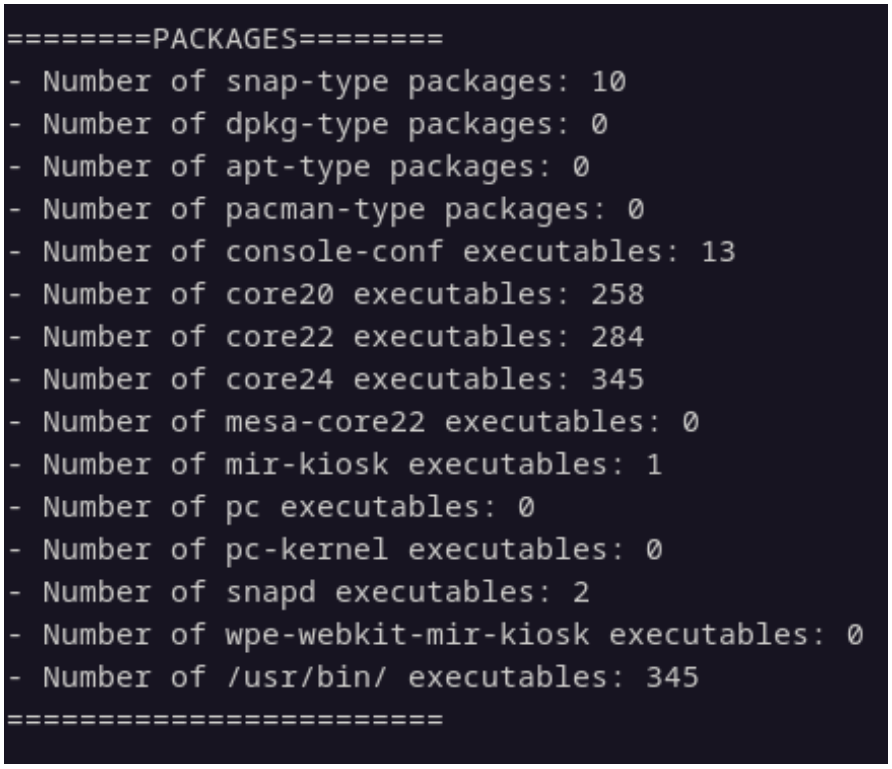

Figura 3.7. Número de paquetes analizados.

Figura 3.8. Analisis del componente OpenSSH.

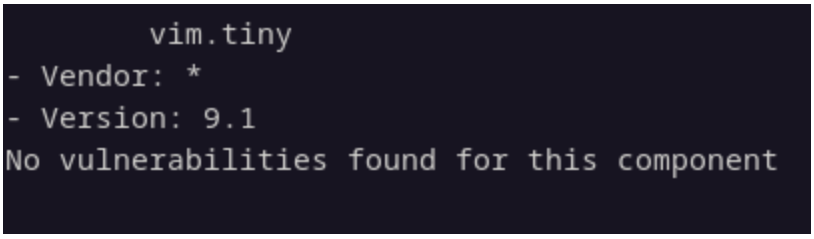

Figura 3.9. Componente vim.tiny.

Aunque se trata de un sistema base, como es Ubuntu Core, y se le instalaron pocos paquetes, se descubre igualmente que hay mucho software vulnerable que podría causar problemas al sistema.

3.5 Limitaciones

Durante la realización de esta herramienta se identificó una primera limitación, y es que no se trata de una herramienta proactiva. Por sí misma no puede descubrir nuevas vulnerabilidades y debe ser ejecutada para poder descubrir las existentes.

Además, durante la creación de esta herramienta se vio que la colección de componentes que venían de gestores de paquetes tenían una facilidad extra ante los ejecutables, ya que los segundos no tenían una salida estandarizada. Esto hace que para obtener la información de estos deban ser llamados uno a uno, y no por medio de una llamada general al sistema de paquetes, probando las distintas opciones que hay para extraer la información de estos componentes.

Aunque el analizador puede ejecutar en cualquier sistema tipo Linux, el mismo está orientado a sistemas Linux IoT, pues cuenta de por sí con muchos menos componentes y muchos menos ejecutables. De hecho, las pruebas hechas en entornos no IoT se han ralentizado mucho debido a la cantidad de llamadas y pruebas que hacía la herramienta para sacar la información de todos los ejecutables. Asimismo, en la sección

de detección de vulnerabilidades de ejecutables se tiene el problema de que para poder sacar la versión de dichos componentes, se tiene que lanzar un comando. En algunos casos, los comandos no necesitan argumentos, y aunque se usen, estos son ignorados y se ejecutan. Tomando como ejemplo, libreoffice-writer, la GUI es lanzada, pero al tener un *timeout* de medio segundo, se mata ese proceso, por lo tanto, a priori no parece un problema. No obstante, hay algunas llamadas que afectan de manera más crítica al sistema y no permiten que el funcionamiento de la herramienta sea la correcta.

Pese a todo esto, la herramienta consigue hacer un completo análisis de los componentes software del sistema, pudiendo ayudar al usuario a identificar los puntos flacos del mismo.

Capítulo 4 - Conclusiones y trabajo futuro

Tras la realización de este TFG se alcanzan las siguientes conclusiones que cumplimentan el trabajo realizado fijado por los objetivos.

En primer lugar, se mostró la importancia de los sistemas Linux para el mundo IoT, se demostraron sus cualidades y la relación que tienen con los dispositivos IoT; esto favorece el auge de Linux en este sector. Posteriormente, se construyó un pequeño sistema basado en Linux, se ejemplificó uno de los usos que puede tener el IoT en el día a día, se mostraron también de manera práctica cómo se montan y los requerimientos que solicitan. Además, sirve a su vez para la demostración de un sistema tipo Linux IoT en acción.

Con todo el entorno de pruebas comentado en el apartado anterior, se desarrolló la herramienta analizadora de componentes software en máquinas Linux, pilar fundamental para este proyecto. Para ello, se investigaron las diversas fuentes que tienen las principales distribuciones Linux para obtener y almacenar software. Esta herramienta, escrita en lenguaje Python, recoge todos los componentes software del sistema e identifica las vulnerabilidades ya conocidas de los mismos, generando un informe, un resumen y destacando aquellas vulnerabilidades que tienen mayor importancia. La herramienta se ejecutó en el Linux que se usó de pruebas. Allí se demostró que, a pesar de que Linux es considerado un sistema sumamente seguro [35] y de que el entorno de pruebas es sencillo, contaba con múltiples vulnerabilidades potenciales.

Con esta información se refuerza la idea de la problemática, en cuanto a seguridad se refiere, que presenta el mundo IoT y que se presentaba en los Capítulos 1 y 2. Por ello, creemos que las revisiones periódicas por medio de herramientas como la desarrollada en este trabajo y la realización de esfuerzos en contar con un sistema

actualizado ayudarán a ir cubriendo los distintos agujeros de seguridad que vayan surgiendo y a ir generando más conciencia sobre la seguridad informática. Tener un sistema seguro es complejo, pero se deben realizar esfuerzos con el fin de garantizar una mejor seguridad.

Concluimos el trabajo con la satisfacción de haber conseguido aquellos puntos propuestos. Queremos, sin embargo, plantear algunos aspectos de mejora como trabajos futuros para la herramienta y así incrementar la seguridad en el mundo IoT.

Una de esas mejoras debería ser el cambio de lenguaje. Si la herramienta estuviera escrita en código C, ganaría en velocidad [36] y generaría análisis más rápidos, punto clave en sistemas con una gran cantidad de paquetes. Además, se puede plantear la herramienta como una ejecución en segundo plano del sistema de arranque, que avise cuando el análisis quede finalizado, de manera que exima al usuario de la obligación de ejecutarlo. Este podría también contar con un historial que notifique solamente aquellas vulnerabilidades nuevas que no hayan sido detectadas hasta el momento. Ambos puntos mejorarían la experiencia del usuario, al tratar con mayor velocidad y limpieza los análisis.

Pese a la importancia que tiene Linux en el sector, existen otros sistemas operativos para IoT, que deberían contar con su propia herramienta de análisis que detalle aquellos puntos débiles detectados dentro del sistema, como realiza el desarrollo Python de este proyecto. Por otra parte, se podría hacer una investigación más exhaustiva en cuanto a los ejecutables, tanto el formato de versionado, como aquellos que pueden afectar a la interfaz gráfica, excluirlos del análisis o buscar una manera de obtener su versión sin necesidad de ejecutar el programa. Esto haría que la herramienta fuera más rápida, y su uso se pudiera expandir a sistemas personales no IoT.

Por último, se podría realizar una plataforma que, dada una vulnerabilidad, reporte aclaraciones o mitigaciones propuestas dando las instrucciones necesarias.

Podrían ir acompañadas de comentarios de otros usuarios que ya hayan podido parchear la vulnerabilidad. Esta información se podría añadir al análisis como información extra para intentar contar con un sistema seguro.

Introduction

The aim of this chapter is to explain the election of this thesis, the scope it has, and the methodology followed.

Motivation

"The Internet of Things (IoT) describes the network of physical objects —"things"— that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the internet."[52] IoT devices range from small sensors and consumer products like smart fridges and coffee machines, to industrial devices like robotic arms. These things can be found in numerous fields and industries [1].

The term was first coined by Kevin Ashton in 1999[53]. Nonetheless, some examples already existed for this kind of device, like the Coca-Cola machine in Carnegie Mellon University. It allowed for a connection to it through the internet, so one could check if there were any drinks left in said machine[54]. Currently, taking into account technological advancements in wireless systems, the number of devices connected to the internet, which were not originally planned to be, has increased. Now we have fridges that can help you make the shopping list [2].

The number of sectors that benefit from IoT is vast and varied: factories, public sector, healthcare, transportation, … This demonstrates the numbers of uses this technology has, and the impact it can have in the world. We consider it relevant to also

[52] https://www.oracle.com/es/internet-of-things/
[53] https://blog.avast.com/es/kevin-ashton-named-the-internet-of-things
[54] https://detri.epn.edu.ec/historia-iot/

speak about IoT from an industrial perspective, as it may allow for improvements in security and manufacturing cycles.

Industrial IoT, known as IIoT, is the collection of sensors, actuators, and devices that work and communicate via the internet with its use centred around industrial applications. This is different to IoT, as its use is more focused on industrial needs, while the latter is used for more general applications[55]. The use cases that IIoT range from machine optimization, to decrease in accidents, occupational or general.

As one can observe in figure 1.1, in 2019 the number of IoT devices connected to the internet was equal to the non-IoT. As we look ahead, for every connected non-IoT device, three IoT devices will also be hooked to the network [3].

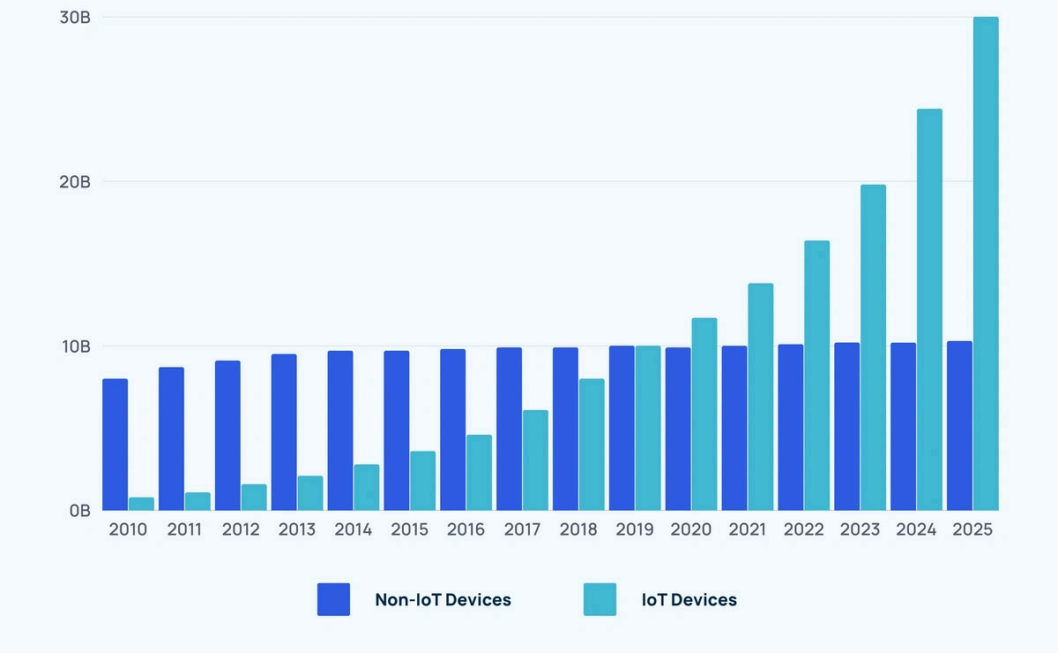

Figure 1.1. Number of global active connections. Source: Hybrid Efficient IDS Against Adversarial Attacks in IoT Networks

The inception of this technology has wrought connectivity and integration to a different number of devices we use in our everyday life. Said devices are low cost, both

[55] https://www.iberdrola.com/innovacion/que-es-iiot

monetarily and energetically, a reason for which both manufacturers and consumers have wagered on this technology. In the following figure, 1.2, one can observe the market growth of IoT devices and how its growth is extrapolated to following years [4].

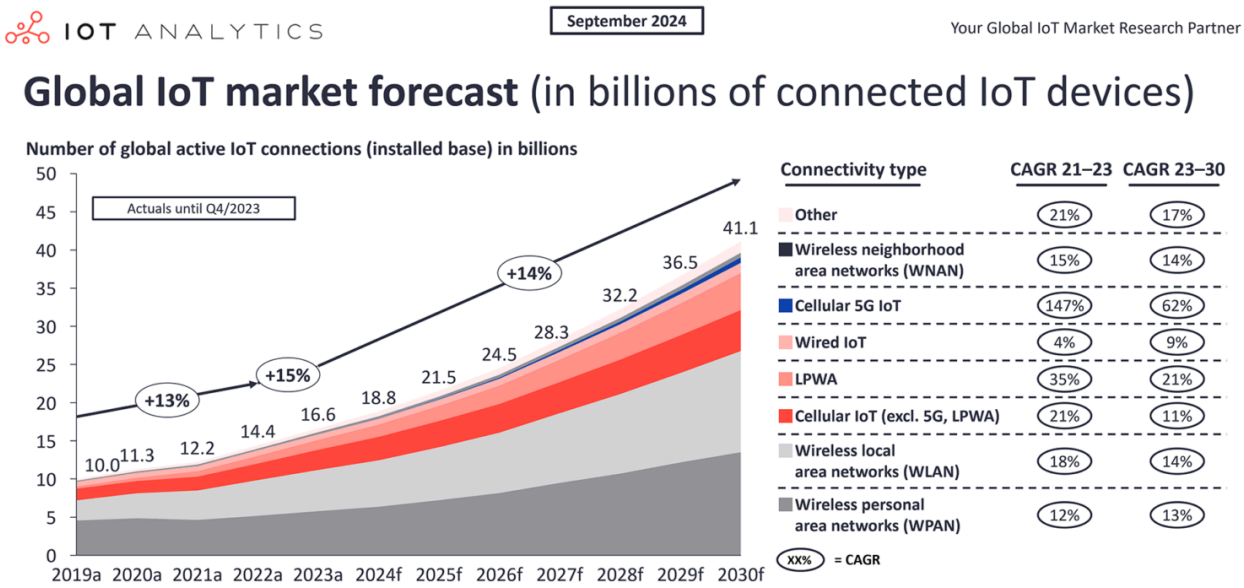

Figure 1.2. Global IoT market forecast. Source: IoT Analytics

This technology has given us many immediate benefits. Nonetheless, it comes bundled with risks associated with the lack of some key security features [5]. If you couple the fact that these devices prioritize a low energy consumption, therefore lowering computational power, and the race to market there exists with these devices, prompts them to not shine for their security. For this reason, many IoT devices show vulnerabilities in authentication protocols, anachronistic firmware and open ports, to name a few [6]. These vulnerabilities present a risk, as cybercriminals can use said security holes to obtain sensitive data, take control of the machine or disable it. This might prove a sizeable problem for consumers that use said devices, aggravating them in critical systems, as are industrial or business scenarios, where a vulnerability can

have grave consequences: halting of trade processes, exposing of sensitive information, large economic loss…

With the current IoT landscape presented above, one can look at the needs this field has in terms of security. Therefore, the goal of this project is to tackle one of the foundational blocks in IoT devices, the operating system. To be precise, in this endeavour we will aim to study and evaluate the different components of a given Linux operating system, creating a simple use case and creating a tool for the analysis. The aforementioned tool will scan the different software components detected in the system and will point to found vulnerabilities that have already been documented.

Objectives

The main objective of this project is to show vulnerabilities found in IoT devices through an analysis of the operating system. Forenamed analysis is going to be centred around a demonstration with the created use case, a kiosk system built on Ubuntu Core. We hope that with this tool, more thought will be put into the security of IoT systems.

Following, the objectives set will be:

- Analyse the involvement of Linux in the IoT landscape.
- The building and configuration of a kiosk-type system on Ubuntu Core.
- Develop a tool that will analyse the components of a Linux system that will detect possible vulnerabilities that affect it.
- Generate an automated analysis, creating a complete report of the system where one can see all vulnerabilities that can impact the system, highlighting those with higher severity.
- Analyse the results obtained from the use of the tool on the kiosk systems created.

Roadmap

The following steps have been followed in order to complete this Final Project:

- Phase 1: Evaluation of current state of the IoT landscape
- Phase 2: Examination of the role Linux has played in the IoT world.
- Phase 3: Creation and configuration of kiosk type system on Ubuntu Core, a Linux distribution geared towards IoT devices.
- Phase 4: Development of a tool that detects software components and the vulnerabilities they may have. Again, centred around IoT Linux systems
- Phase 5: Perform tests of the created tool on the created kiosk. Taking data and drawing conclusions
- Phase 6: Write-up of this document, recording everything done

Conclusions and future work

From the above sections, we arrive at the following conclusions. Firstly, the importance of Linux systems for the current IoT landscape was highlighted, showing its strengths and the relationship with IoT devices.

Following the above, a small Linux based system was created, serving as an example of one of the uses it can have, providing the step-by-step process for building it and the requirements. This also serves as a testing platform for the tool created.

With the environment built, the tool was created for Linux systems, a foundational pillar for this dissertation. For this, an investigation was conducted on the different package managers that see common use in different distributions. The tool, written in Python, obtains all software components in the systems and identifies vulnerabilities that have already been identified and catalogued in services like CVE or NVD. Both a detailed report and a summary are created, highlighting the vulnerabilities that may have a bigger impact on the system. Later the tool was used to analyze the system created above, and although Linux systems are considered highly secure [37] and the environment created was rather simple, many vulnerabilities were found.

With the information above, the ideas presented in chapter 1 and 2, regarding the lack of security and the number of problems in IoT systems, are reinforced. Therefore we believe that a tool like ours, used periodically to analyze the system and keeping the software of these machines updated, would help in keeping the systems secure and increase awareness of security holes.

We finish this project with the satisfaction of having been able to achieve the goals presented. Nonetheless, we find relevant thinking about things that could be done to improve the work presented in possible future iterations of this tool.

One of the first improvements would be to change the programming language. If the tool was written in C, execution times would see an improvement [38], generating analysis in less time, key for systems with a high volume of packages and executables. Secondly, we could change the program's function so that instead of running it manually, it could be run in the background while the system is starting, notifying when the execution is completed. History could also be implemented so that only new vulnerabilities found would be documented. Both these points would improve ease of use, making the analysis faster and cleaner.

Furthermore, although Linux plays a large role in IoT systems, there are other operating systems for which our tool would not prove usable. Further study would be needed in order to be able to obtain application information for these other operating systems. The tool could also be improved further by investigating in more detail how the different standards for version information can be found in the executables. Moreover, having a list of the executables that affect the display in some way, for systems with a GUI, would prove helpful in analysing these components without having to execute them.

On a final note, the creation of a platform that, given a vulnerability, could give mitigations to said problems giving clear instructions would broaden the scope of this tool. Not only signalling out the vulnerabilities, but also showing ways of dealing with them. Said vulnerabilities could be accompanied by comments created by other users that have been able to mitigate the issue. This information could be added to the analysis as a supplement to try to achieve a more secure system.

Contribuciones personales

Pablo García Fernández

Cabe destacar que el enfoque inicial del proyecto era otro distinto, pero que pretendía también invitar a la reflexión sobre el estado de la seguridad en el mundo IoT. Por ello se mantuvo la investigación previa sobre el mundo IoT y la seguridad en el mismo. En esa búsqueda de información sobre el tema me centré en la lectura y comprensión de aquellos estudios que descubren y/o explotan distintas brechas de seguridad en sistemas IoT. Este paso me introdujo en el mundo de la documentación en cuanto a las vulnerabilidades se refiere. Ahí me informé de los distintos mecanismos que existen para su formalización, identificando las características de la misma, con el *string vector*, y por medio de ellas evaluarla. Siguiendo esta línea de trabajo también aprendí sobre las debilidades, que pueden ayudar a prevenir futuras vulnerabilidades. Todo este aprendizaje me sirvió para el correcto desarrollo del proyecto y de la memoria, incluyendo en este documento la información esencial al respecto para la comprensión del mismo.

Tras un estudio que realizamos sobre la importancia de los sistemas Linux en el ecosistema IoT y tras cruzarnos con varios sistemas operativos decidimos trabajar sobre el sistema de Ubuntu Core, por lo que me puse manos a la obra a tratar de entender cómo funcionaba. Leyendo la documentación y consultando alguna que otra fuente de información me fuí guiando para configurarlo a nuestro gusto. Para ello tuve también que conocer QEMU, la herramienta que usamos para virtualizar el sistema. Conocer la configuración me sirvió para plasmar esa información en el proyecto, dejando constancia del paso a paso seguido que fuí realizando.

Durante el proceso, y como parte del anterior planteamiento del proyecto, encontré vulnerabilidades ya conocidas que podrían afectar a Ubuntu Core, en concreto

a su inicio de sesión, que funciona por SSH. Gracias a eso pudimos encauzar el posterior desarrollo Python con la certeza de que se estaban encontrando las vulnerabilidades que realmente afectan al sistema y que no contábamos con información falsa.

En la parte del desarrollo comencé, tras un boceto que consultaba sobre un componente concreto, con la estructura que iba a seguir la herramienta. Para ello modulé distintas funciones que iban ayudando al paso a paso. Cabe destacar la función constructora del CPE, identificador ya comentado que sirve para consultar a la API de NVD, y las distintas funciones encargadas de recoger los componentes software instalados por un gestor de paquetes. Todas esas partes, junto a la recogida de componentes que no vienen por un gestor de paquetes realizada por mi compañero Alejandro, finalizaban en la consulta directa a la API, donde tras los resultados se realizaba, por medio de otra función creada, un informe sobre las vulnerabilidades. Este punto es clave para nuestra herramienta, ya que brinda la información sobre el estado del sistema al usuario. Con el fin de poder mejorar esa información, decidí desarrollar una funcionalidad que, en base a una serie de criterios, identificaba si la vulnerabilidad era destacable, para así ser notificada al usuario.

Por último, todos estos puntos tratados fueron completados con mi compañero, plasmando las ideas y tareas en este documento, trabajando especialmente en aquellas que había tratado de primera mano. Conjuntamente a los tutores confirmamos el correcto estado de esta memoria, finalizando así este proyecto que a mí me sigue confirmando el vulnerable estado en el que se encuentra el mundo IoT.

Alejandro Luque Villegas

En cuanto a mi aportación a este trabajo, lo primero que me gustaría comentar es que en un principio el planteamiento era completamente distinto. Nuestro objetivo era usar un dispositivo similar al FlipperZero[56] llamado EvilCrowRF-V2[57] para realizar un análisis topológico de la red y posteriormente conectarnos con la API de CVE para evaluar que posibles vulnerabilidades se encontraban en la misma. Desafortunadamente, el uso de un firmware no oficial causó el *brickeo* del dispositivo y tras hablar con el creador del mismo, nos dimos cuenta de que esperar a otro dispositivo no sería factible y, por tanto, hemos acabado realizando el trabajo que precede estas palabras.

En cuanto a mi aportación al proyecto, la primera tarea fue la investigación y búsqueda de noticias para plasmar la necesidad de este trabajo y la falta de conciencia que hay de la falta de seguridad en los dispositivos IoT. La gran mayoría de artículos plasman que muchos de los problemas que tienen estos sistemas son humanos, ya que no se suelen cambiar las claves por defectos por desconocimiento o desgana, dejando abierto un vector de ataque claro y fácil de explotar.

Con esta tarea también me di cuenta de la cantidad de ataques que se pueden llevar a cabo gracias a la infección de dispositivos IoT. Una débil seguridad en un ecosistema IoT no solo genera problemas en el entorno, sino que, se podría llegar a usar un número de dispositivos infectados generando una botnet para llevar a cabo ataques de Denegación de Servicios masivos.

El siguiente paso en el que contribuí fue en la investigación de los distintos sistemas operativos que se pueden llegar a usar en dispositivos IoT y la factibilidad de su posible virtualización. Para ello tomé apuntes usando una plataforma que me

[56] https://flipperzero.one/
[57] https://github.com/joelsernamoreno/EvilCrowRF-V2

permite crear hipervínculos en un formato que puedo compartir fácilmente con mi compañero Pablo. Esta herramienta se llama Obsidian y permite crear notas enlazadas, la cual nos ayudó mucho a ver qué sistemas podrían tener similitudes en cuanto a la manera de virtualizar o mecanismos internos que podrían funcionar de una manera similar. Muchos de estos sistemas se quedaron en la fase de investigación, ya fuera por imposibilidad de virtualización, problemas a la hora de virtualizar o directamente estar detrás de un muro de pago. Esto nos llevó a trabajar con dos sistemas, Ubuntu Core y Windows 11 LTSC Enterprise Edition, este último descartado por problemas con virtualización y búsqueda de vulnerabilidades relevantes.

Después de conseguir virtualizar Ubuntu Core con la ayuda de Pablo, empecé a buscar información sobre CVE, NVD y las APIs correspondientes, creando un script muy sencillo con el que hacer *requests* a la página correspondiente. Pablo extendió este script creando un método de creación de *CPE string* y obteniendo información de los distintos paquetes del sistema.

Tras algunas pruebas, no obtuvimos los resultados que esperábamos y tuvimos que indagar un poco más. Aquí es cuando encontramos los ejecutables de los distintos paquetes *snap* y pensamos que la mejor manera de obtener sus versiones era probando con los distintos *flags* de consulta. De esta manera, cada ejecutable mostraba su versión de una manera distinta. La única manera que encontré para poder estandarizar los datos de versión fue haciendo una lista de las distintas maneras en las que se obtenía la información de versión y usar expresiones regulares para poder capturar el número de versión y el nombre del binario para poder hacer consultas a la base de datos CVE.

Bibliografía

[1] Watters, A. (2023, July 18). Top 30+ iot statistics and facts you should know for 2023. *Global Technology Industry Association*. https://gtia.org/blog/top-internet-of-things-stats-facts

[2] Jha, V. (2025, January 4). *Samsung's ai-powered fridge takes grocery shopping to the next level with instacart integration - Aitechtonic*. AiTechtonic - Informative & Entertaining Text Media. https://aitechtonic.com/samsungs-ai-powered-fridge/

[3] Aljbour, E., & Al-Haija, Q. A. (2024, January 1). *Hybrid efficient IDS against adversarial attacks in iot networks*. Springer Nature Switzerland. https://link.springer.com/chapter/10.1007/978-3-031-69986-3_7#Sec5

[4] Sinha, S. (2024, September 3). *Number of connected IoT devices growing 13% to 18.8 billion*. IoT Analytics. https://iot-analytics.com/number-connected-iot-devices/

[5] Dabrowska, M. (2024, March 26). *Security concerns in IoT: Addressing the challenges head-on*. IoT Now News - How to Run an IoT Enabled Business. https://www.iot-now.com/2024/03/26/143458-security-concerns-in-iot-addressing-the-challenges-head-on/?cn-reloaded=1

[6] Spektor, H. (2024, April 19). *Top 10 iot vulnerabilities and how to mitigate them*. Sternum IoT. https://sternumiot.com/iot-blog/top-10-iot-vulnerabilities-and-how-to-mitigate-them/

[7] Said, F. (2024, December 9). *CWE vs CVE: Diferencias clave explicadas*. Xygeni | Software Supply Chain Security. https://xygeni.io/es/blog/cwe-vs-cve-key-differences-explained/

[8] Susnjara, S., & Smalley, I. (n.d.). Sistema operativo. *IBM*. Retrieved August 15, 2025, from https://www.ibm.com/es-es/think/topics/operating-systems

[9] Nivollet, D. (2024, August 5). *Tipos de sistemas operativos: Cuáles son y para qué sirven*. iFP. https://www.ifp.es/blog/tipos-de-sistemas-operativos

[10] Susnjara, S., & Smalley, I. (n.d.-b). Sistema operativo en tiempo real (RTOS). *IBM*. Retrieved August 15, 2025, from https://www.ibm.com/es-es/think/topics/real-time-operating-system

[11] Flinders, M., & Smalley, I. (n.d.). Linux. *IBM*. Retrieved August 15, 2025, from https://www.ibm.com/es-es/think/topics/linux

[12] Galicia, F. P. L. (2024, August 27). *Guía sobre qué es Linux y cómo funciona*. GoDaddy Resources - LATAM. https://www.godaddy.com/resources/latam/desarrollo/linux-que-es

[13] Reviriego, J. F. (2024, July 17). *¿Qué es el sistema Linux y cuáles son sus ventajas?* Blog de SEAS. https://www.seas.es/blog/informatica/que-es-el-sistema-linux-y-cuales-son-sus-ventajas/

[14] SAKHRI, M. (2025, April 6). *14 best operating systems for iot devices in 2025 (open source & RTOS options)*. Tech2Geek. https://www.tech2geek.net/14-best-operating-systems-for-the-internet-of-things-iot/

[15] Fuentes, F. (2023, April 24). *¿Qué es un gateway IoT y por qué es importante?* Arsys. https://www.arsys.es/blog/gateway-iot

[16] Parada, M. (2021, March 1). Qué es un paquete en Linux y qué gestores existen. *OpenWebinars.Net*. https://openwebinars.net/blog/que-es-un-paquete-en-linux-y-que-gestores-existen/

[17] Mershad, K., & Cheikhrouhou, O. (2023). Lightweight blockchain solutions: Taxonomy, research progress, and comprehensive review. *Internet of Things*, *24*, 100984. https://doi.org/10.1016/j.iot.2023.100984

[18] Solis, D. C. (2022, December 29). Virtualización de servidores: Ventajas y desventajas. *OpenWebinars.Net*. https://openwebinars.net/blog/virtualizacion-de-servidores-ventajas-y-desventajas/

[19] Industrial, E. S. (2022, July 15). *Ventajas y desventajas de las máquinas virtuales - Axentio*. Axentio - Innovación a Través de La Tecnología. https://www.axentio.com/ventajas-y-desventajas-de-las-maquinas-virtuales/

[20] GeeksforGeeks. (2024, September 27). Internet of things with python. *GeeksforGeeks*. https://www.geeksforgeeks.org/python/internet-of-things-with-python/

[21] Goodwin, M. (n.d.). API. *IBM*. Retrieved August 15, 2025, from https://www.ibm.com/think/topics/api

[22] Marugán, M. (2022, November 14). ¿Qué es una API? Tipos de API y ejemplos. *Cyberclick*. https://www.cyberclick.es/que-es/api-tipos-de-api-y-ejemplos

[23] Bonaventura, D., Esposito, S., & Bella, G. (2023). Smart bulbs can be hacked to hack into your household. *Proceedings of the 20th International Conference on Security and Cryptography*, 218–229. https://arxiv.org/pdf/2308.09019

[24] Daws, R. (2025, January 22). *Mirai IoT botnet powers record 5.6 Tbps DDoS attack*. Internet of Things News. https://iottechnews.com/news/mirai-iot-botnet-powers-record-ddos-attack/

[25] Bella, G., & Biondi, P. (2021, November 20). *You overtrust your printer*. arXiv.Org. https://arxiv.org/abs/2111.10645

[26] Intelligence, M. T. (2024, May 30). *Exposed and vulnerable: Recent attacks highlight critical need to protect internet-exposed OT devices*. Microsoft Security Blog. https://www.microsoft.com/en-us/security/blog/2024/05/30/exposed-and-vulnerable-recent-attacks-highlight-critical-need-to-protect-internet-exposed-ot-devices/

[27] Mejía-Granda, C. M., Fernández-Alemán, J. L., Carrillo-de-Gea, J. M., & García-Berná, J. A. (2023). Security vulnerabilities in healthcare: An analysis of medical devices and software. *Medical & Biological Engineering & Computing*, *62*(1), 257–273. https://doi.org/10.1007/s11517-023-02912-0

[28] Coker, J. (2024, June 10). IoT vulnerabilities skyrocket, becoming key entry point for attackers. *Infosecurity Magazine*. https://www.infosecurity-magazine.com/news/iot-vulnerabilities-entry-point/

[29] Constantin, L. (2025, August 11). Robots are just as plagued by security vulnerabilities as IoT devices. *Computerworld*. https://www.computerworld.com/article/1682274/robots-are-just-as-plagued-by-security-vulnerabilities-as-iot-devices-2.html

[30] RedSwitches. (2024, August 12). *Snap package: The ultimate guide*. Medium. https://medium.com/@redswitches/snap-package-the-ultimate-guide-c32dafb0bbfd

[31] Canonical. (2020, February 19). 19th February 2020 – Canonical today announced that Bosch Rexroth has selected Ubuntu Core for their app-based platform ctrlX AUTOMATION. ctrlX AUTOMATION leverages Ubuntu Core, designed for embedded devices, and snaps, the universal Linux application containers, to deliver an open source platform to remove the barriers between machine co […]. *Ubuntu*. https://ubuntu.com/blog/bosch-rexroth-adopts-ubuntu-core-and-snaps-for-app-based-ctrlx-automation-platform

[32] Pastor, J. (2018, August 17). Debian cumple 25 años, y estas son las razones por las que ha sido crucial para la historia de Linux. *Xataka*. https://www.xataka.com/servicios/debian-cumple-25-anos-estas-razones-que-ha-sido-crucial-para-historia-linux

[33] Vaughan-Nichols, S. (2016, June 14). Ubuntu Snap takes charge of Linux desktop and IoT software distribution. *ZDNET*. https://www.zdnet.com/article/ubuntu-snap-takes-charge-of-linux-desktop-and-iot-software-distribution/

[34] Gaucher, A. (2023, September 22). Cadena SER. *Cadena SER*. https://cadenaser.com/comunitat-valenciana/2023/09/22/los-autobuses-urbanos-de-alicante-dispondran-de-un-servicio-de-informacion-con-pantallas-en-su-interior-radio-alicante/

[35] Kaspersky. (2018, November 21). *¿Es seguro Linux?* /. https://www.kaspersky.es/resource-center/definitions/linux

[36] apalaciper1. (n.d.). *Diferencias Lenguaje C y Python (Aitor)*. Retrieved August 15, 2025, from https://iesbenjamin.educacion.navarra.es/blogs/tic2/2025/02/10/diferencias-lenguaje-c-y-python/